CEPC NOTE

CEPC-SIMU-2017-001

October 20, 2017

中国科学院高能物理研究所
Institute of High Energy Physics
Chinese Academy of Sciences

Full Simulation Software at CEPC

CEPC Software Group

Abstract

The detector optimisation and physical analysis are being performed through full detector simulation at CEPC. As quick starting, a ILD-used simulation framework is selected. According the progress of CEPC, this simulation tool is also developed as a new version. This note will introduce the framework and its usage. And the development will also be described.

E-mail address: fucd@ihep.ac.cn

Contents

1 Introduction

A project on a high energy Circular Electron Positron Collider (CEPC) as a Higgs and/or Z factory and a subsequent Super proton-proton Collider (SPPC) [1] is at the stage of Conceptual Design Report (CDR) study. For the detector design study, performance analysis based on a full simulation is an important method. In order to perform the study, a simulation tool based on Object Modeling for compact calorimeters (**M**odellierung mit **O**bjekten eines **K**ompakten **Ka**lorimeters) [2] is developed, which has been apply on the International Linear Collider detector (ILC/ILD) [3, 4, 5] study. The new developed version of Mokka at CEPC is also called as MokkaC.

2 Framework of the software for full simulation

A Geant4-based [6] full simulation framework has been developed first for the ILC/ILD. In this framework, some prepared geometry-drivers for sub-detectors are used to setup the geometries of sub-detectors, and they are assembled as the whole large spectrometer. And then the transportation of all input particles will be simulated by Geant4. Finally, information of obtained hits responded in the sub-detectors are outputted as a LCIO-architectural file [7].

2.1 Framework of framework

The Mokka includes four primary interfaces: generator, geometry, output and controller. Its kernel classes include CGAGeometryManager, VSensitiveDetector, VSubDetectorDriver, VSuperSubDetectorDriver, MySQLWrapper, SteppingAction, CGASteppingAction, MyPlacement, VHit and its derived classes TRKHit, TRKFTDHit, TPCHit and CalHit, as shown in Figure 1. While performing a simulation, the processes in Mokka is described as shown in Figure 2.

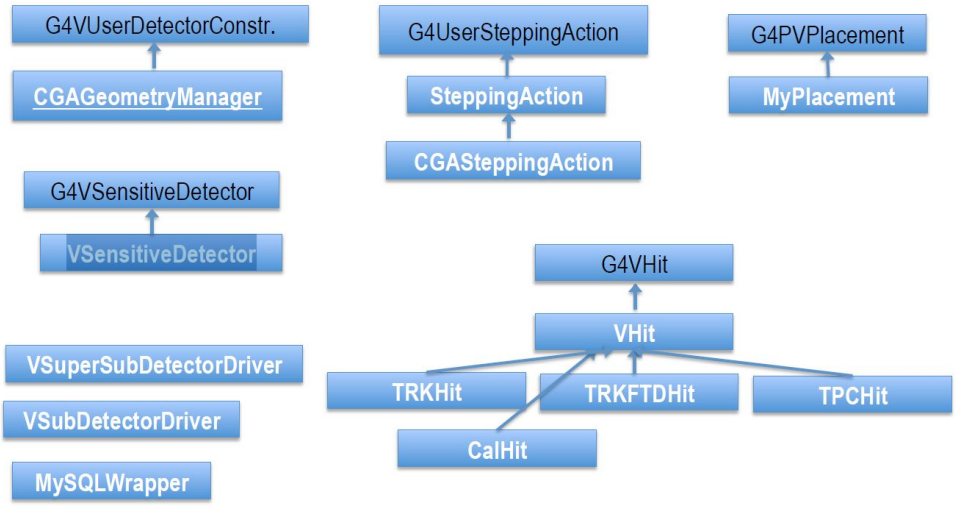

Figure 1: The Mokka kernal class

The simulating events can be input through files with several certain formats (stdhep, hepevt, pairs) which are produced by physical generators such as WHIZARD[8], besides Geant4-supported generators and their extension such as particleGun and gps. For file format, Mokka will read and unpack it into G4PrimaryParticle and G4PrimaryVertex as the beginning of G4Event, and into MCParticle of LCIO as one of output at the same time.

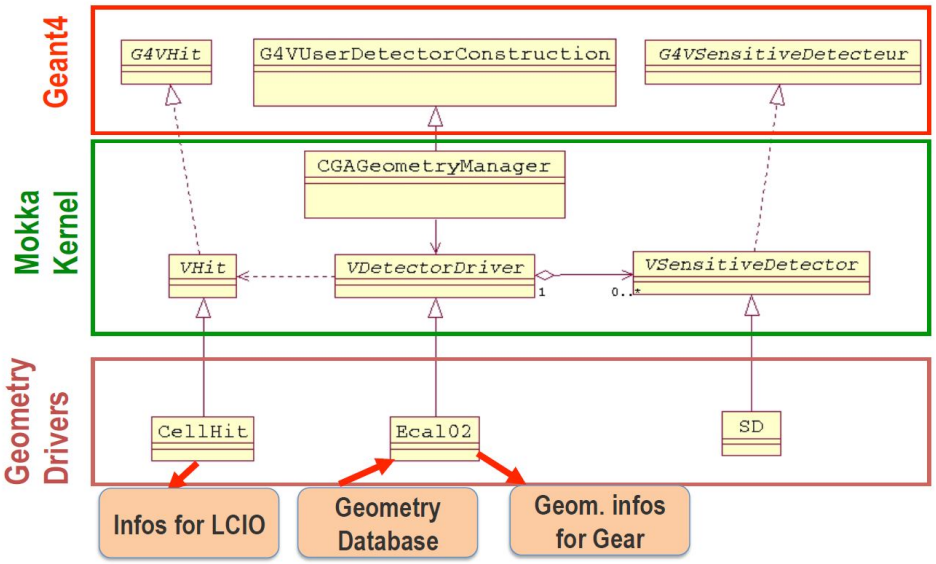

Figure 2: The interior relationship between Mokka's modules and Geant4

Geant4 can complete simulating the particle transportation in detector, once its geometry is told. Most work of Mokka framework is to achieve this goal. The geometry is managed by CGAGeometryManager through VDetectorDriver-based drivers. Once user-defined drivers are built, Mokka can assembly them to a whole detector according to an optional model. The model is setup in database, which includes sub-detector ingredients, sub-detectors' driver setup and parameters. More details can refer to [10][11]

2.2 Guide for development

There are two ways for development of Mokka: kernel and geometry. As shown in Figure 3, The geometry directory includes CGA as geometry manager, MokkaGear as geometry feed-back interface for reconstruction and others for drivers. In general, user need only modify the drivers and their setup. Therefore, this section will only introduce how to write a driver and how to build a detector model.

In fact, Mokka has prepared two base classes: VSubDetectorDriver for driver and VSuperSubDetectorDriver for supper driver. The driver is regarded as construction of one sub-detector or more, while the supper driver is used as interface to transfer parameters of sub-detector to the corresponding driver temporarily, since it is inefficient to modify the parameters frequently while a sub-detector has not been confirmed. Therefore, one sub-detector ingredient is composed of one driver, or one driver plus one supper driver. The detail will be described in the section 2.3. So the key is to write a class inherited from VSubDetectorDriver or VSuperSubDetectorDriver. Exampling by VSubDetectorDriver, the driver ExampleDriver include the constructor ExampleDriver::ExampleDriver(), another necessary generic function ExampleDriver::ContextualConstruct(const CGAGeometryEnvironment& env, G4LogicalVolume* worldLog) and one optional function ExampleDriver::GearSetup(), whose head file is shown in Figure 4. There are two names in the driver, one is the driver name and another is the prefix name of file as ASCII output for hits. In ContextualConstruct function, the parameters can be obtained through CGAGeometryEnvironment& env, and then are used to construct the sub-detector like as normal Geant4 construction. It is noted that there are three important points listed as following:

- env.GetDBName() to obtain the database name for this driver, and then read parameters from the database. By this method, we can keep the geometry parameters outside the code.

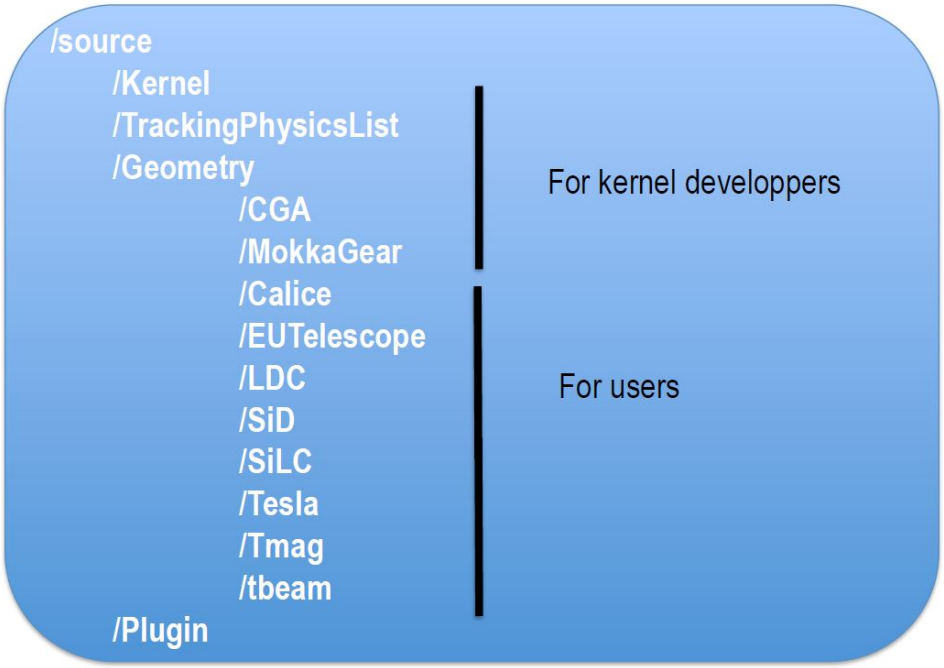

Figure 3: The code's structure of Mokka

- worldLog is the pointer of the World volume created in CGAGeometryManager.
- Sensitive detector should be also setup to the logical volume with sensitive material, and RegisterSensitiveDetector(SD) is necessary, where SD is the pointer of the sensitive detector class inherited from the class VSensitiveDetector.

```
//
#ifndef ExampleDriver_h
#define ExampleDriver_h 1

#include "VSubDetectorDriver.hh"
//#include needed head file

class G4LogicalVolume;

class ExampleDriver : public VSubDetectorDriver{
public:
  ExampleDriver() : VSubDetectorDriver("ExampleDriver","AsciiHitsOutput"){}
  ~ExampleDriver();

  G4bool ContextualConstruct(const CGAGeometryEnvironment &env, G4LogicalVolume *worldLog);

#ifdef MOKKA_GEAR
  void GearSetup();
#endif
  // addtional function

private:
  // data member needed

};
#endif
```

Figure 4: The code's structure of Mokka

For sensitive detector, existing classes such as TRKSiSD00 for track hit and SDHcalBarrel for calorimeter hit are good referable examples.

Once all drivers are ready, the left work is to build a detector model in database. Currently, the database often used is **models03**. Figure 5 shows how to obtain the setup of the detector model through

the database. The database **models03** includes these tables as listed in Table 1. Using the key name of model, the Mokka framework will find out the name of detector concept in the table **model**, and the sub-detectors list in the table **ingredients**. According to the name of detector concept, the world size and the region sizes of tracker and calorimeter will be known through searching the table **detector_concept**, and they are used to define the world volume and the physical region. To search the sub-detectors list in the table **sub_detector**, it is clear to know the building elements of a sub-detector, database, driver and subdriver. If the subdriver is not blank, it means that a VSuperSubDetectorDriver driver will be called before the driver. The VSuperSubDetectorDriver driver will build a temporary database to feed parameter values to the normal driver. Otherwise, the driver will read parameters from the declared database directly and build the geometry.

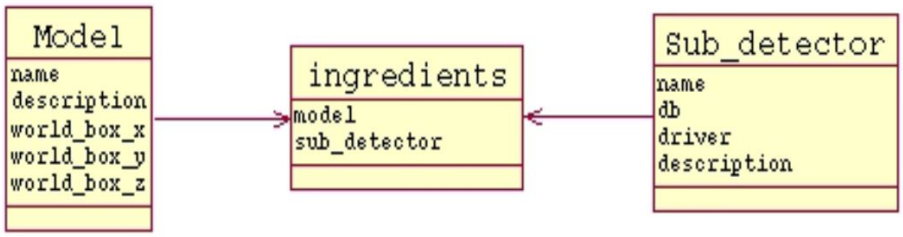

Figure 5: The database models03

Table 1: The tables in the database **models03**.

Tables in models03	Table contents
detector_concept	name, description, world_box_hx, world_box_hy, world_box_hz, tracker_region_rmax, tracker_region_zmax, calorimeter_region_rmax, calorimeter_region_zmax
ingredients	id, model, sub_detector, build_order
model	name, description, detector_concept, model_status
model_parameters	model, parameter, default_value
parameters	name, description, default_value
scripts	name, script
setup	name, description
setup_parameters	setup, parameter, value
sharing	driver, parameter, driver_default_value
sub_detector	id, name, db, driver, description, subdriver
tmp_databases	name, connection, time_stamp

The parameters in Mokka database have three versions: **parameters**, **model_parameters** and **sharing**. Their relationships are shown in Figure 6. The **parameters** include all parameters in database. They will be overwritten by the **sharing** values corresponding to the driver, and then the **model_parameters** values. It is noticed that the parameter will keep the previous value if it does not exist at this step. Then the parameters are still possibly overwritten by the values in the steering file if any, and finally by the environment values if a driver changes the values, which will affect on those following drivers, but not previous drivers.

2.3 Usage of simulation software

Since the Mokka framework is Geant4-based, the Geant4 should have been installed certainly.

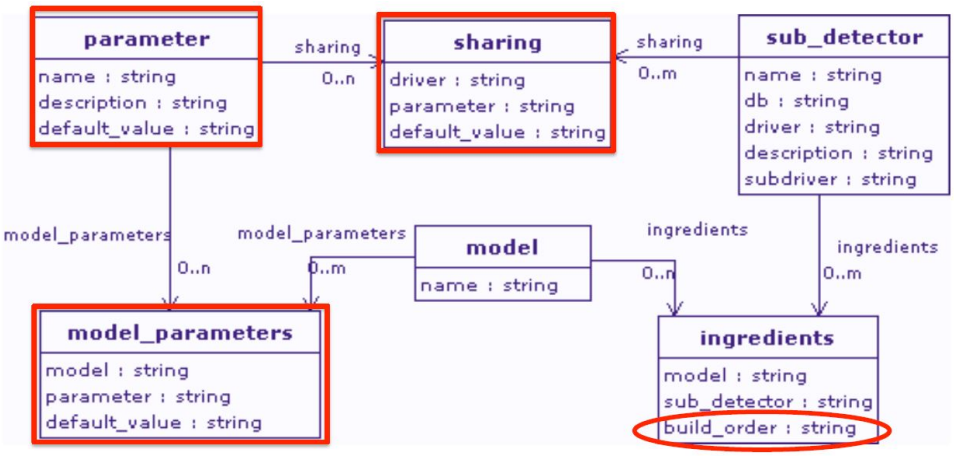

Figure 6: Parameters in Mokka database.

2.3.1 Installation

The Mokka can run together with the ILC software. Generally, the environment script for the ILC software already includes the Mokka environment:

```
export MOKKA="/afs/ihep.ac.cn/soft/common/gcc/v01-17-05/Mokka/mokka-08-03"
export PATH="$MOKKA/bin:$PATH"
```

Sometimes, we need to modify the codes of Mokka. The original codes can be copied from the AFS server, and also downloaded through GIT [9] server (http://cepcgit.ihep.ac.cn/cepcsoft/Mokka). Therefore, a modified Mokka will storage at a directory different with the default, for example $WORKDIR/MokkaC/MokkaC-00-01. And then the new Mokka environment can be set through:

```
export MOKKA="$WORKDIR/MokkaC/MokkaC-00-01"
export PATH="$MOKKA/bin:$PATH"
```

Here the searching directories for dynamic library are not set for Mokka, the reason is that the directory of dynamic library has been designated while compiling. Therefore, it is important to check the compiling option. If ”-Wl,-rpath” has been removed in option, the following command should be added accordingly:

```
export LD_LIBRARY_PATH="$MOKKA/lib:$LD_LIBRARY_PATH"
```

2.3.2 Compile

After copying or downloading the Mokka codes into a work directory, there are two methods to compile. One is to use cmake package, another is to use the Geant4 GNUmakefile and compile directly.

For cmake case, the steps are:

```
cd $MOKKA
mkdir build
cd build
cmake -C $ILCSOFT/ILCSoft.cmake ..
make
make install
```

For GNUmakefile case, the steps are:

```
cd $MOKKA/source
make
make install
```

2.3.3 Running a Mokka simulation

After installation, it is simple to run Mokka by the command:

```
Mokka [-option] steering.macro
```

where available options are listed in Table 2, and a simple steering file is examples as:

```
/Mokka/init/BatchMode true
/Mokka/init/printLevel 0
/Mokka/init/detectorModel CEPC_v1

/Mokka/init/dbHost 202.122.37.75
/Mokka/init/user consult
/Mokka/init/dbPasswd consult

/Mokka/init/lcioFilename     cepc.slcio
/Mokka/init/initialMacroFile event.macro

/Mokka/init/lcioDetailedShowerMode true
/Mokka/init/userInitBool WriteCompleteHepEvt true
/Mokka/init/lcioWriteMode WRITE_NEW
/Mokka/init/lcioStoreCalHitPosition true
/Mokka/init/lcioDetailedTRKHitMode TPCCollection
```

In steering file, there are a series of control command. The command is always followed by one or more blank-separated string like as /Mokka/init/detectorModel CEPC_v1. The available commands are listed in following, and their usages and descriptions are also described. According to requirement, they can be added into the steering file.

- /Mokka/Draw bool, to set drawFlag for end of events (default is false).
- /Mokka/LoadEvent integer, to load a simulated event (depending on the persistency mode).
- /Mokka/init/BatchMode bool, no interactive session if in batch mode. Just executes a given macro file, if any.
- /Mokka/init/detectorModel string, the detector model to be used as defined in the models database.
- /Mokka/init/modelsDBName string, the models database name on the MySQL server to be used.
- /Mokka/init/materialsDBName string, the models database name on the MySQL server to be used.
- /Mokka/init/detectorSetup string, the detector setup to be used as defined in the models database.
- /Mokka/init/dbHost string, the host machine where the MySQL server is running.
- /Mokka/init/user string, the user name to connect to the MySQL server.

Table 2: The available options in Mokka command.

-H	print options list
-m <filename>	specifies a macro file to be executed before running
-l <filename>	specifies the filename for LCIO output
-o <dirname>	specifies the directory for ASCII output
-v	specifies that the -o option above points to an old output directory with data just to be reloaded for visualisation (no physics in this case)
-P	specifies NOT to save event primaries (event primaries are saved by default in persistent mode)
-T	specifies NOT to save primary trajectories (primary trajectories are saved by default in persistent m
-c <double>	specifies the Geant 4 production range cut in mm
-C <double>	specifies the range cut in the sensitive materials in mm
-B <double>	specifies a magnetic field factor
-t <double>	specifies the TPC primary energy cut in MeV (enables the user to control the TPC output file lengt
-b	specifies BRAHMS backward facility. (generates GEANT3 code)
-M <model>	specifies the detector model to be simulated
-s <setup>	specifies the detector setup to be simulated
-S <subdet>	specifies to build only one subdetector (overrides the -M option)
-h <hostname>	specifies the hostname of the MySQL server
-u <user>	specifies the MySQL user
-p <password>	specifies the MySQL password
-e <pdgfile>	specifies pdg file with extra particles (e.g. particles.tbl)
-r <seed>	specifies the random seed. Overides any value specified in the steering file
-n <runNumber>	specifies the mcRunNumber. Overides any value specified in the steering file
-N <runNumber>	specifies the DataRunNumber. Overides any value specified in the steering file
-g <filename>	specifies the filename for Gear geometry output
-G <filename>	specifies filename for XML that merges into regular output (merges dominantly into output)

- /Mokka/init/dbPasswd string, the password to connect to the MySQL server.
- /Mokka/init/subDetector string, to specifies just a sub detector name to be built.
- /Mokka/init/initialMacroFile string, name of the initial macro file to be executed after startup.
- /Mokka/init/outDirName string, name of the output directory - this implies ASCII output!
- /Mokka/init/lcioFilename string, name of LCIO output file - this implies LCIO output!
- /Mokka/init/lcioEventParameter string, parameter added to every LCIO event - usage: type[int,float,string] name value.
- /Mokka/init/lcioWriteMode string, write mode of LCIO output file: "WRITE_APPEND" or "WRITE_NEW".
- /Mokka/init/pythiaFilename string, name of PYTHIA input file.
- /Mokka/init/dumpG3 bool, to specifies BRAHMS backward facility (generates Geant3 Fortran code).
- /Mokka/init/savingTrajectories bool, to specifies whether to save primary trajectories (default is true), Only if "/init/outDirName" is set (ASCII mode).
- /Mokka/init/savingPrimaries bool, to specifies whether to save primaries. default is true), Only if "/init/outDirName" is set (ASCII mode).

- /Mokka/init/visumode bool, to specifies whether to start in visualization mode, Only if "/init/outDirName" is set (ASCII mode).
- /Mokka/init/BFactor float, to specifies a magnetic field factor (0 to 1).
- /Mokka/init/userDeltaIntersection float, to changes delta intersection parameter of the field manager (mm).
- /Mokka/init/userDeltaOneStep float, to changes DeltaOneStep parameter of the field manager (mm).
- /Mokka/init/rangeCut float, to specifies the production Geant4 range cut (mm).
- /Mokka/init/pcbRangeCut float, to specifies the production Geant4 range cut in the PCB (mm).
- /Mokka/init/radiatorRangeCut float, to specifies the production Geant4 range cut in the radiator (mm).
- /Mokka/init/activeRangeCut float, to specifies the production Geant4 range cut in the active material (mm).
- /Mokka/init/TPCCut float, to specifies the TPC primary energy cut.
- /Mokka/init/TrackingPhysicsListMSOn bool, turns on Multiple Scattering in the TrackingPhysicsList, default is ON.
- /Mokka/init/TrackingPhysicsListELossOn bool, turns on EM Energy Loss in the TrackingPhysicsList, default is ON.
- /Mokka/init/TPCLowPtCut float, to Specifies the Pt threshold above which hits are produced using the pad-ring doublet volumes in the TPC. Below this value of Pt, hits will only be produced if TPCLowPtStepLimit is true. In which case a step limiter will be employed for these low Pt tracks, and a hit created once TPCLowPtHitSeparation is exceeded.
- /Mokka/init/TPCLowPtStepLimit bool, turns on the creation of hits caused by tracks with Pt less than TPCLowPtCut, and activates the Pt dependant Step Limmiter in the TPC. default is false.
- /Mokka/init/TPCLowPtStoreMCPForHits bool, turns on the storing of MCParticles for hits caused by tracks with Pt less than TPCLowPtCut, default is false.
- /Mokka/init/TPCLowPtMaxStepLength float, to specifies the Max step length for the Pt dependant Step Limmiter.
- /Mokka/init/TPCLowPtMaxHitSeparation float, to specifies the max distance between those hits with pt less than TPCLowPtCut.
- /Mokka/init/registerPlugin string, activate the plugin with the given name to be executed at runtime, and all plugins will be called in the order they have been activated.
- /Mokka/init/physicsListName string, to specify the name of the physics list to be used for the simulation.
- /Mokka/init/userInitString string string, define a user variable of type string (first as name, second as value).

- /Mokka/init/userInitDouble string double, define a user variable of type double (first as name, second as value, optional third as unit).
- /Mokka/init/userInitInt string int, define a user variable of type int (first as name, second as value).
- /Mokka/init/userInitBool string bool, define a user variable of type bool (first as name, second as value).
- /Mokka/init/lcioDetailedShowerMode bool, if true, LCIO file will contain detailed MC contribution from secondaries in calorimeter showers.
- /Mokka/init/useOldHEPLCIO bool, if true, we use the old scheme of reading hepevt/stdhep files - debug option.
- /Mokka/init/FixStdHepLeptonFSR bool, if true, a fix is applied when reading in stdhep files, that avoids double counting of leptons in the case of FSR on the lepton. Default value is on.
- /Mokka/init/lcioWriteCompleteHepEvt bool, if true, the output LCIO file will contain the complete list of particles present in the event file with their original generator (HepEvt) status preserved.
- /Mokka/init/lcioWriteParentsForBackscatter bool, if true, the output LCIO MCParticle collection will have shower particles stored if they are parents of back scattered particles. This is the old (pre DBD) behaviour.
- /Mokka/init/CONFIG_ANGLE float, to specify the configuration angle for test beam setups.
- /Mokka/init/lorentzTransformationAngle float, to specify the angle for the Lorentz transformation of primary particles.
- /Mokka/init/primaryVertexSpreadZ float, to specify the spread of z for primary vertex.
- /Mokka/init/globalModelParameter string string, define a global parameter for the geometry model (first as name, second as value).
- /Mokka/init/startEventNumber int, begins the run from the given event number.
- /Mokka/init/printLevel int, sets the verbosity level of Mokka.
- /Mokka/init/randomSeed int, sets the random seed of the CLHEP generator. Overidden by command line option -r <seed>.
- /Mokka/init/pairParticlesPerEvent int, number of pair particles to be simulayted in on (LCIO) event with GuineaPig pair background files.
- /Mokka/init/lcioDetailedTRKHitMode string, enable momentum information in tracker hits for a given hits collection.
- /Mokka/init/detailedHitsStoring string, store all hits in the sensitive volume.
- /Mokka/init/lcioStoreCalHitPosition bool, to specifies whether to save CalHit position in LCIO mode. (default is true).
- /Mokka/init/EditGeometry/addSubDetector strings, to add a Sub-Detector to the current Detector Model in a given build-order.

- /Mokka/init/EditGeometry/rmSubDetector string, to remove a subdetector (or all) from the current Detector Model.
- /Mokka/init/MokkaGearFileName string, set file name for Geometry xml output file.
- /Mokka/init/MokkaGearMergeSource string, set file name for xml file. It will dominantly merge with regular MokkaGear output.
- /Mokka/init/mcRunNumber int, sets the Monte Carlo run number. Overidden by command line option -n <runnumber>.
- /Mokka/init/PDGFile string, sets the particle data for HepPDT, probably called particle.tbl.
- /Mokka/Visu/Refresh, refresh the view.
- /Mokka/Visu/Detector/Mode, set the rendering mode for a given sub detector and deep.
- /Mokka/Visu/Detector/Colour, Set the rendering color for a given sub detector deep.
- /Mokka/Visu/Detector/Daughters bool, set the daughter's visibility for a given sub detector and deep.
- /Mokka/Visu/Detector/Visibility, set the visibility for a given sub detector.
- /Mokka/Visu/Detector/ListGeometryTree, prints the sub detector names, visibility and sub detector trees.
- /Mokka/Visu/Detector/ImmediateMode, automatical refresh of the viewer after each command.
- /Mokka/Visu/Detector/Reset, reset the vis attributes to the model default.
- /Mokka/Visu/Detector/DumpGDML, dumps the GDML for all detector or for a given sub detector Logical Volume.
- /Mokka/Visu/Detector/DumpVRML, dumps the VRML for all detector or for a given sub detector.

On the other hand, the commands on generator are input generally in the file designated by /Mokka/init/initialMacroFile. For a prepared file by generators, its events can be input easily by the command /generator/generator <file> like as:

```
/generator/generator test.stdhep
/run/beamOn 100
```

This example will read 100 events from test.stdhep (Binary file in HEPEvt common block format). For HEPEvt (ASCII file in reduced HEPEvt common block format) or pairs (ASCII file in Guinea Pig format) input file, the command is

```
/generator/generator test.HEPEvt
```

or

```
/generator/generator test.pairs
```

As described in previous section, particleGun or gps is also supported by Mokka. The following is an example for particleGun.

```
/generator/generator particleGun
/gun/position 0 0 0 mm
/gun/direction 1 0 0
/gun/directionSmearingMode uniform
/gun/thetaSmearing 90 deg
/gun/phiSmearing 180 deg
/gun/momentum 70 GeV
/gun/momentumSmearing 60.0 GeV
/gun/momentumSmearingMode uniform
/gun/particle e-
/run/beamOn 100
```

In this example, the particleGun generator will produce one electron uniform in 4π space and uniform from 10 GeV to 130 GeV. Besides uniform distribution, particleGun supports gaussian smearing mode too, and its usage is like as

```
/gun/momentum 125 GeV
/gun/momentumSmearing 5.0 GeV
/gun/momentumSmearingMode gaussian
```

It denotes to produce a particle whose momentum is 125 GeV and spread for momentum 5 GeV. The smearing for direction is similar.

For gps, the usage is similar with particleGun and there are more options. More details for options can refer to [12].

3 Development at CEPC

Because the original Mokka framework depends on the MySQL database, it is not convenient to modify the detector design for detector optimisation. In order to avoid to change the database frequently, some development on Mokka have been done at CEPC. Therefore, the new Mokka is called as MokkaC (Mokka at **C**EP**C**). Comparing MokkaC with Mokka, the dominant upgrades include support for new sub-detector, HepMC and slcio events interface, updated drivers and more drivers.

3.1 Command support for new sub-detector

In Mokka framework, a driver is used to implement the geometry into Geant4 only if it has been built in database. A new command /Mokka/init/EditGeometry/newSubDetector is added to ignore the requirement. So MokkaC will allow a database-independent driver to implement the geometry. At the same time, avoid saving parameters in codes, the parameters for this type of driver will be input through files, and the steering file is one of choices.

The usage of new command is like as:

```
/Mokka/init/EditGeometry/newSubDetector <NAME> <ORDER> [DRIVER] [DB]
```

where NAME is the name of sub-detector, ORDER is the built order in sub-detectors list, DRIVER is the name of driver and DB is the name of database. If DRIVER and DB are not set, a NAME same driver will be used.

3.2 HepMC and slcio interface

Because pre-selecting processor will output events as .slcio file (LCIO) before simulation, a slcio interface is needed. On the other hand, HepMC is also a event format often used by generator. After adding them into generator support, the all supported generators are listed in Table 3.

Table 3: The MokkaC supported generator

.stdhep
.HEPEvt or .hepevt
.slcio
.pairs
.hepmc
particleGun
gps

3.3 Updated driver

The updated drivers and new drivers are listed in Table 4.

Table 4: The updated drivers and new drivers in MokkaC

Driver	Status	Description
Tube_cepc	updated	Add new type of pipe for doubly crossing beam pipe
SHcalRpc01	updated	Add support for scintillator as sensitive material
yoke05	updated	Available to modify through steering file
SiCal	new	Simple driver flexible to modify layer structure and material
SiTracker01	new	A driver for silicon-based tracker flexible to modify layer structure
GeneralInterface	new	A general driver to quickly implement if a Geant4 normal constructor is ready

For the both drivers SiCal and SiTracker01, their parameters are input through steering file. Exampled by SiTracker01, Figure 7 shows how to input parameters into the driver SiTracker01. In parameter SiTrackerLayerStructure, the semicolons separate each sub-detector type such as VXD, SIT, FTD_PIXEL and FTD_STRIP. And in each sub-detector, the sub-parameters are like as:

```
TYPE,A:a,B:b,...
```

where A, B are materials of sub-layer, and a, b are their thickness. If the thickness is negative, it denotes the corresponding sub-layer is sensitive. Similar in parameter SiTrackerBarrel and SiTrackerEndcap, the semicolons separate each layer. And in each layer, the sub-parameters are:

```
TYPE, radius, half length in Z, number of ladders
```

for barrel; and

```
TYPE, rin, rout, z position, number of petal
```

for endcap.

4 Brief introduction of the CEPC detector model

The main detector models are set in database which are called as CEPC_v1, CEPC_v2, etc... and users don't need to modify them generally. If it is necessary sometime, the modification can be input through the steering file. This section describes some main models and how to change them.

```
/Mokka/init/globalModelParameter SiTrackerBarrel VXD,15.9,78,10;VXD,25,125,10;VXD,36.9,150,11;VXD,\
38,150,11;VXD,57.9,175,17;VXD,59,175,17;VXD,153,368,10;VXD,156,368,10;SIT,321,644,19;SIT,324,644,1\
9;SIT,603.4,920,38;SIT,606.4,920,38;SIT,1000,1380,62;SIT,1003,1380,62;SIT,1410,1840,89;SIT,1413,18\
40,89;SIT,1811,2300,115;SIT,1814,2300,115

/Mokka/init/globalModelParameter SiTrackerEndcap FTD_PIXEL,30,150.76,220,16;FTD_PIXEL,50.6,150.76,\
371,16;FTD_STRIP,70.98,325,644,16;FTD_STRIP,110,611,920,16;FTD_STRIP,176,1004,1380,16;FTD_STRIP,23\
4,1414,1840,16;FTD_STRIP,293,1815,2300,16

/Mokka/init/globalModelParameter SiTrackerLayerStructure SIT,Si:-0.15,Si:0.0024,Peek:0.1,CarbonFib\
er:0.08,Rohacell50D:0.9,Epoxy:0.08,CarbonFiber:0.08;FTD_PIXEL,Si:-0.2,Si:0.0048,CarbonFiber:0.16,R\
ohacell50D:1.8,Peek:0.2;FTD_STRIP,Si:-0.15,Peek:0.2,CarbonFiber:0.16,Rohacell50D:1.8,Epoxy:0.175,C\
arbonFiber:0.16,Si:0.0048,Si:-0.15;VXD,Si:-0.05,kapton:0.05,aluminium:0.01,SiC_foam:0.94
```

Figure 7: Input parameters to the driver SiTracker01 through globalModelParameter.

4.0.1 Baseline

There are two main detector model to simulate till now, CEPC_v1 and CEPC_v4. CEPC_v1 is a model for ILD-like detector. The dominant difference between CEPC_v1 and CEPC_v4 is in the design of Machine Detector Interface. The beam pipe of CEPC_v1 is single pipe, and that of CEPC_v4 is doubly-pipe with crossing angle. Table lists their main parameters and differences between CEPC_v1 and CEPC_v4.

Table 5: Differences between CEPC_v1 and CEPC_v4. R denotes the radius, and Z the half length for barrel or the Z position for endcap.

Sub-detector		CEPC_v1	CEPC_v4
MDI	Type	single pipe	doubly pipe
Lcal	R	60 mm - 172 mm	30 mm - 100 mm
VXD	R/Z	16 mm/62.5 mm, 37 mm/125 mm, 58 mm/125 mm	same as CEPC_v1
FTD	Z (mm)	220, 371.3, 645, 846, 1057.5	220, 371.3, 643, 844, 925
SIT, SET	R	153 mm, 300 mm, 1811 mm, 1813.5 mm	same as CEPC_v1
TPC	R†	384 mm - 1718 mm	same as CEPC_v1
	Z	2350 mm	
Ecal	R	1843 mm - 2028 mm	1843 mm - 2028 mm
	Z	2450 mm - 2635 mm	2450 mm - 2635 mm
	Cell size	5.0833 mm	10.1667 mm
	R‡	226.8 mm	245 mm
Hcal	R	2058 mm - 3385.53 mm	2058 mm - 3143.43 mm
	Cell size	10.408 mm	10.407 mm
	layer number	48	40
Yoke	R§	4415 mm (B), 300 mm (E)	4174 mm (B), 300 mm (E)
Field		3.5 T	3.0 T

† Sensitive region
‡ Inner radius of endcap
§ Inner radius, B for barrel and E for endcap

4.0.2 CEPC_v1 with doubly pipe

How to simulate with the model CEPC_v1 has been described in Section 2.3.3. If only modifying a little, it can be done through the steering file. Based on CEPC_v1, to change single pipe to doubly pipe, the following steering file can be referred.

```
#remove old MDI and add new
/Mokka/init/EditGeometry/rmSubDetector tube_cepc
```

```
/Mokka/init/EditGeometry/rmSubDetector mask_cepc
/Mokka/init/EditGeometry/newSubDetector new_tube 901 Tube_cepc TMP_DB03_33
/Mokka/init/EditGeometry/newSubDetector new_mask 902 Mask_cepc TMP_DB04_33
#change the position of Lcal
/Mokka/init/globalModelParameter Lcal_inner_radius 30
/Mokka/init/globalModelParameter Lcal_outer_radius 100
/Mokka/init/globalModelParameter Lcal_z_begin 930
/Mokka/init/globalModelParameter Lcal_z_thickness 130
#change FTD
/Mokka/init/EditGeometry/rmSubDetector ftd_cepc
/Mokka/init/EditGeometry/newSubDetector SiTracker01 1
/Mokka/init/globalModelParameter SiTrackerEndcap FTD_PIXEL,29.5,147.57,220,16;FTD_PIXEL,3
147.57,371,16;FTD_STRIP,32.52,292.56,645,16;FTD_STRIP,33.98,302.87,846,16;FTD_STRIP,34.57
302.87,925,16
/Mokka/init/globalModelParameter SiTrackerLayerStructure FTD_PIXEL,Si:-0.02,CarbonFiber:1
FTD_STRIP,Si:-0.2,CarbonFiber:2,Si:-0.2
```

There are three steps: first remove old MDI (tube and mask) and add new MDI, second change the Lcal, last change the FTD. Since the new MDI overlaps with the Lcal and the FTD in CEPC_v1 model, the second and third steps are also indispensable. Otherwise, it is possible to cause unreliable results by overlap, after changing the MDI.

4.0.3 Modification on Ecal

```
/Mokka/init/globalModelParameter Ecal_Si_thickness 1
/Mokka/init/globalModelParameter Ecal_nlayers1 10
/Mokka/init/globalModelParameter Ecal_nlayers2 10
/Mokka/init/globalModelParameter Ecal_Sc_Si_mix 0000000000
```

4.0.4 Modification on VXD

Available modification on VXD through the steering file is to change the densities of the support layers and the sensitive layers like as:

```
/Mokka/init/globalModelParameter VXDSupportScale 2.0
/Mokka/init/globalModelParameter VXDSiliconScale 1.0
```

Based on the default materials, their densities are changed by the scale factors. In a way, the modification of density is equivalent to change the thickness of layer. Therefore, it is possible to study the influence on track performance by material budget to optimize the vertex detector.

4.0.5 Modification on Yoke

The modification of Yoke can be made as following:

```
/Mokka/init/globalModelParameter YokeUserLayer true
/Mokka/init/globalModelParameter YokeGapThickness 25
/Mokka/init/globalModelParameter YokeIronThickness 100
/Mokka/init/globalModelParameter YokeLayerNumber 12
/Mokka/init/globalModelParameter YokeBarrelEndcapGap 5
```

It is noticed that the modification will not work if YokeUserLayer is false, in order to avoid unexpect variation by code update. This is different as described in Section 2.1.

4.0.6 Modification on Hcal

The default hadronic calorimeter is compose of radiator material and Resistive Plate Chamber (RPC) and its sensitive material is RPC gas. The RPC can replaced with scintillator, and then the sensitive material will be polystyrene. In following example, the scintillator is 3 mm thick and covered with 0.5 mm S235 iron on one side and 0.7 mm PCB and 0.1 mm copper on another side.

```
/Mokka/init/globalModelParameter Hcal_sensitive_model scintillator
/Mokka/init/globalModelParameter Hcal_scintillator_thickness 3
/Mokka/init/globalModelParameter Hcal_steel_cassette_thickness 0.5
/Mokka/init/globalModelParameter Hcal_Cu_thickness 0.1
/Mokka/init/globalModelParameter Hcal_PCB_thickness 0.7
```

4.0.7 Modification on TPC

```
/Mokka/init/globalModelParameter TPC_Ecal_Hcal_barrel_halfZ 2350
/Mokka/init/globalModelParameter TPC_inner_radius 329
/Mokka/init/globalModelParameter TPC_outer_radius 1808
/Mokka/init/globalModelParameter TPC_pad_height 6
/Mokka/init/globalModelParameter TPC_pad_width 1
/Mokka/init/globalModelParameter TPC_max_step_length 10
```

5 Validation

The public validation tool is being developed, and simple validating methods for simulation are geometry viewer and hits distribution.

5.1 CEPC_v1

The over viewer of the CEPC_v1 detector in simulation is shown in Figure 8.

5.2 VXD and SIT

The X-Y viewers of the VXD and the SIT in the CEPC_v1 detector are shown in Figure 9. The left is the VXD and the SIT. The right is the VXD only. The green part is the end-plate for support and the electronics.

5.3 Hcal

A example of hits in the Hcal is shown in Figure 10.

5.4 MDI

Different with CEPC_v1, a new MDI is designed in CEPC_v4. This new MDI has a doubly crossing pipe as shown in Figure 11. In order to check whether the geometry is rightly integrated, a size-scaled and angle-enlarged pipe is tested as shown in Figure 12. It is noticed that the design of MDI is continuing to update.

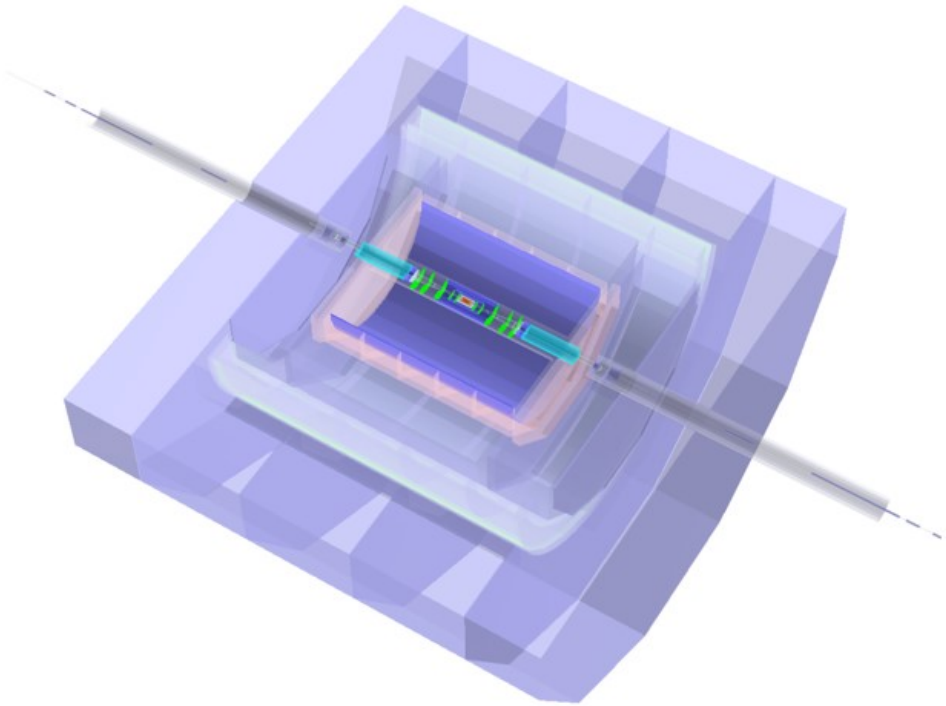

Figure 8: The geometry of CEPC_v1 detector in simulation.

5.5 Full silicon-based tracker

A full silicon-based tracker is also designed at CEPC to replace the TPC. Its geometry is shown in Figure 13, and an example of its hits is shown in Figure 14.

5.6 Dual readout calorimeter

The dual readout calorimeter (DRC) is integrated by the driver GeneralInterface and a Geant4 general construction. A tower-reduced DRC is built for check as shown in Figure 15. The real DRC is constructed by 283 tower in phi and 52 barrel tower and 40 endcap twoer in theta. The fibers in tower are shown in Figure 16.

Figure 17 and Figure 18 show the hits of the DRC and the hits in tower system for 1000 single particles, respectively.

Figure 19 and Figure 20 show some distributions of the fiber ID and the deposited energy for single muon and single electron, respectively.

References

[1] M. Ahmad et al. (The CEPC-SPPC Study Group), CEPC-SPPC: Preliminary Conceptual Design Report Volume I - Physics & Detector, IHEP-CEPC-DR-2015-01/IHEP-EP-2015-01/IHEP-TH-2015-01

[2] P. Mora de Freitas and H. Videau, Detector simulation with Mokka/GEANT4: Present and future, LC-TOOL-2003-010, 623-627 (2002)

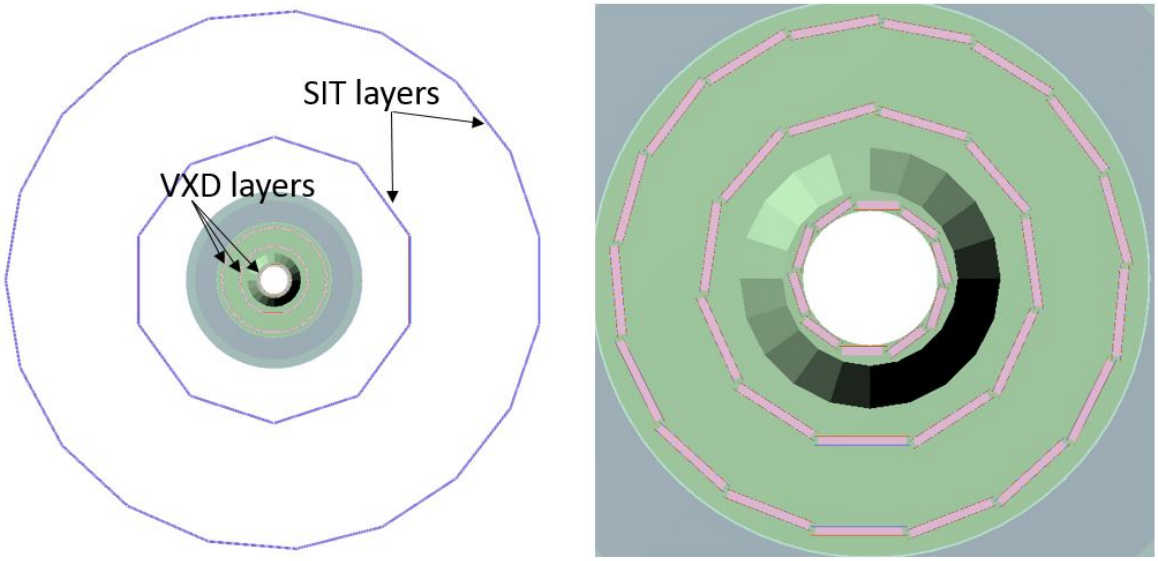

Figure 9: The geometry of the VXD and the SIT sub-detectors.

[3] T. Behnke et al., The International Linear Collider Technical Design Report C Volume 1: Executive Summary, arXiv:1306.6327 [physics.acc-ph]

[4] T. Abe et al. (ILD Concept Group), The International Large Detector: letter of Intent, arXiv:1006.3396 [hep-ex]

[5] T. Behnke et al., The International Linear Collider Technical Design Report C Volume 4: Detectors, arXiv:1306.6329 [physics.ins-det]

[6] http://geant4.cern.ch/

[7] http://lcio.desy.de/v02-09/doc/manual_html/manual.html

[8] http://whizard.hepforge.org/

[9] https://git-scm.com/doc

[10] https://llrforge.in2p3.fr/viewvc/Mokka/trunk/doc/CGADoc/CGAIndex.html

[11] https://llrforge.in2p3.fr/viewvc/Mokka/trunk/doc/Kernel/geometry_drivers.html

[12] http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch02s07.html

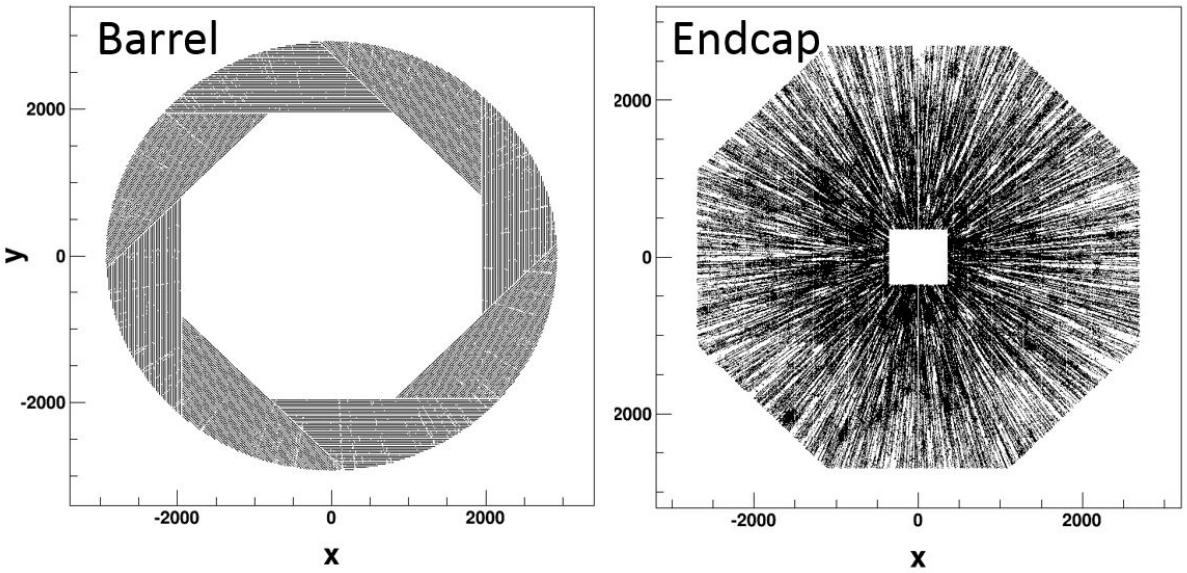

Figure 10: The hits in Hcal for 100,000 single muon particles.

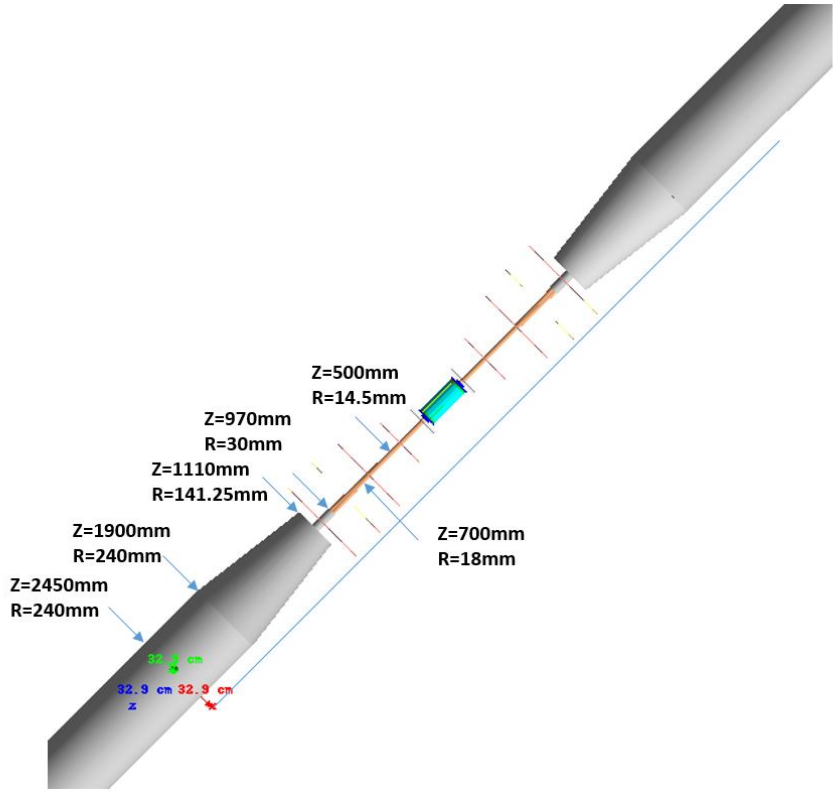

Figure 11: The geometry of the MDI in CEPC_v4.

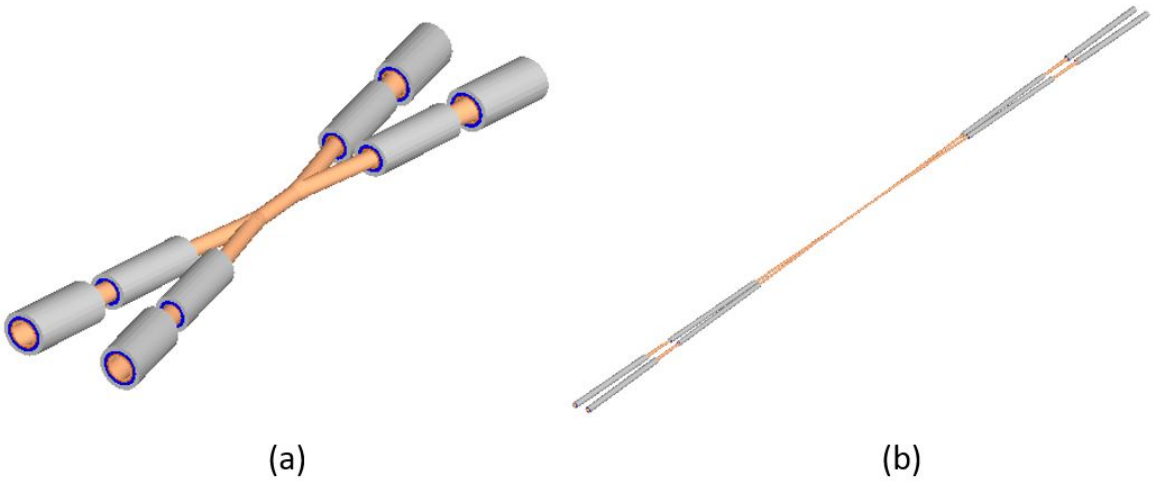

Figure 12: The geometry of the beam pipe in CEPC_v4. (a) a size-scaled and angle-enlarged pipe and (b) the designed pipe in CEPC_v4.

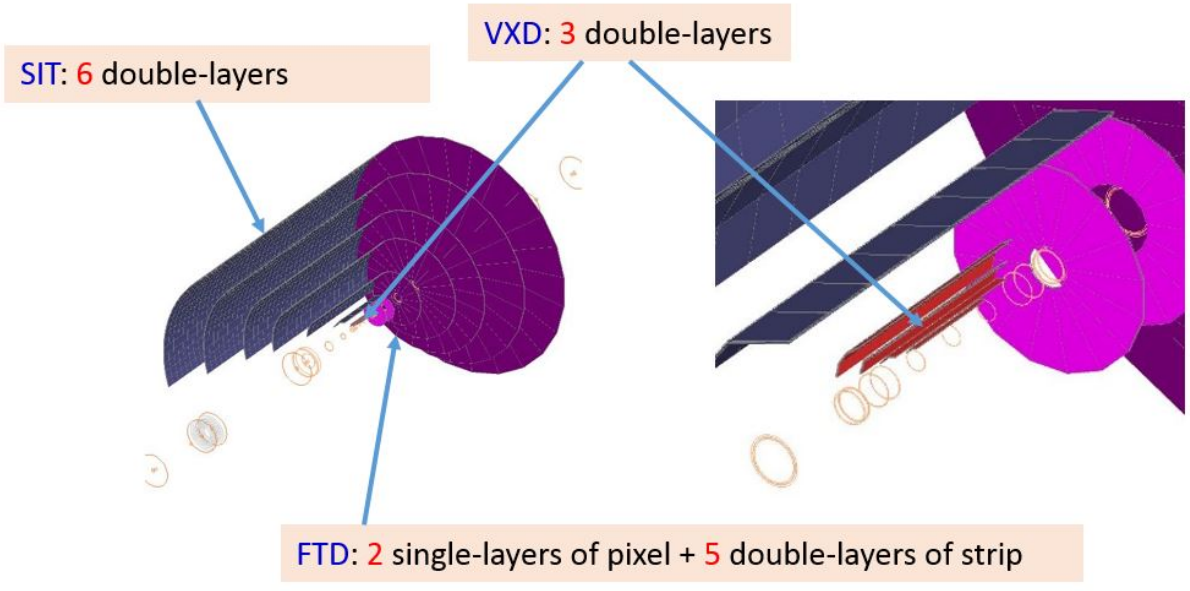

Figure 13: The geometry of the full silicon-based tracker at CEPC.

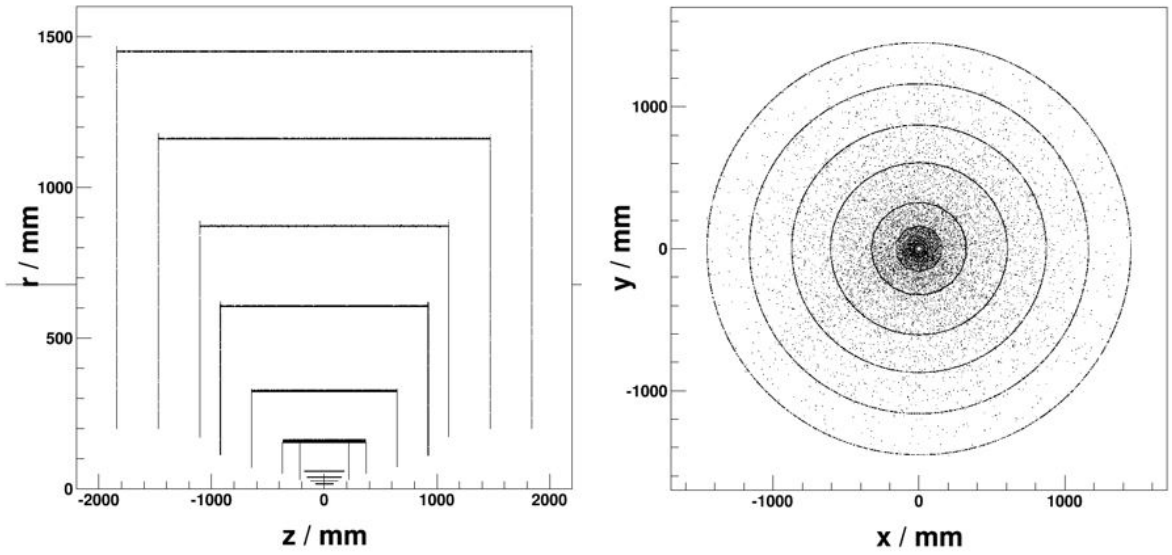

Figure 14: The hits in the full silicon-based tracker for single muon.

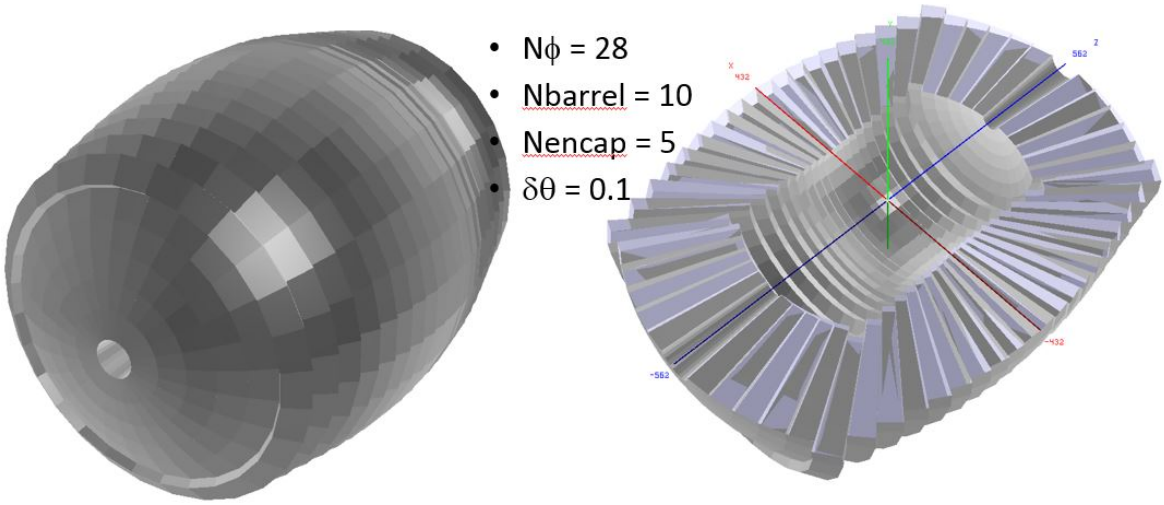

Figure 15: The tower-reduced geometry of DRC.

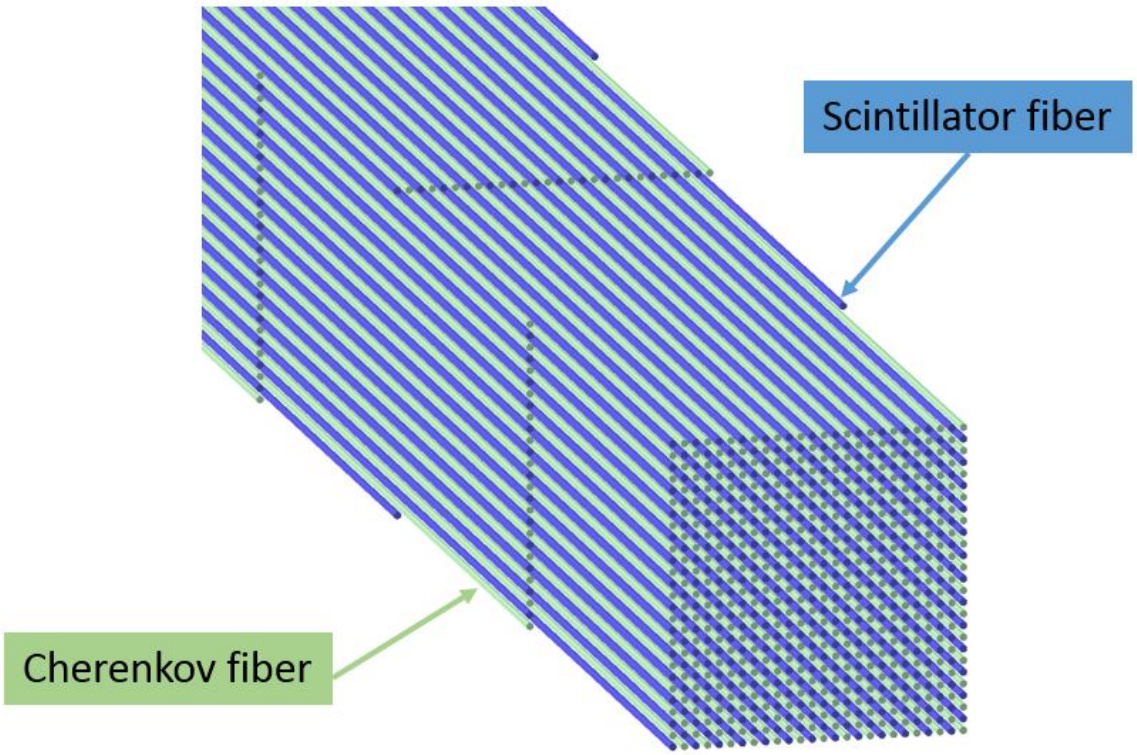

Figure 16: The fibers in tower.

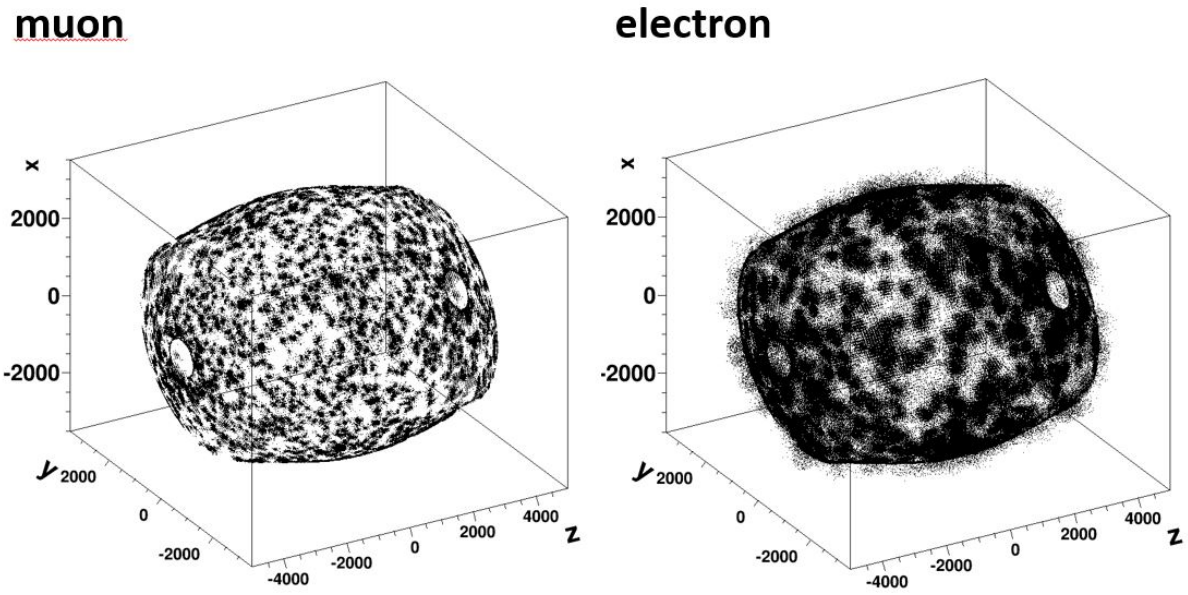

Figure 17: The hits of the DRC for 1000 single particles.

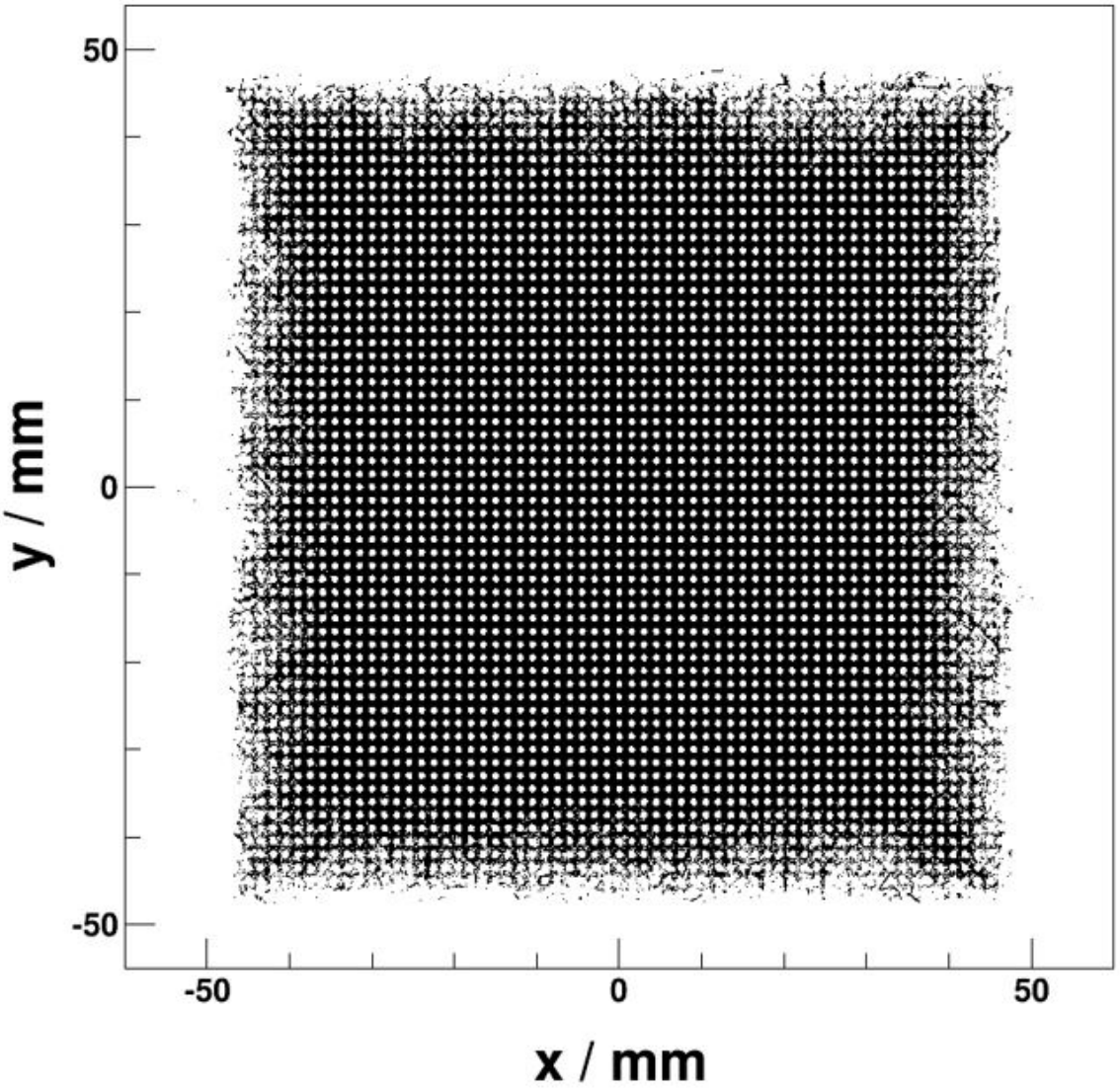

Figure 18: The hits in tower system.

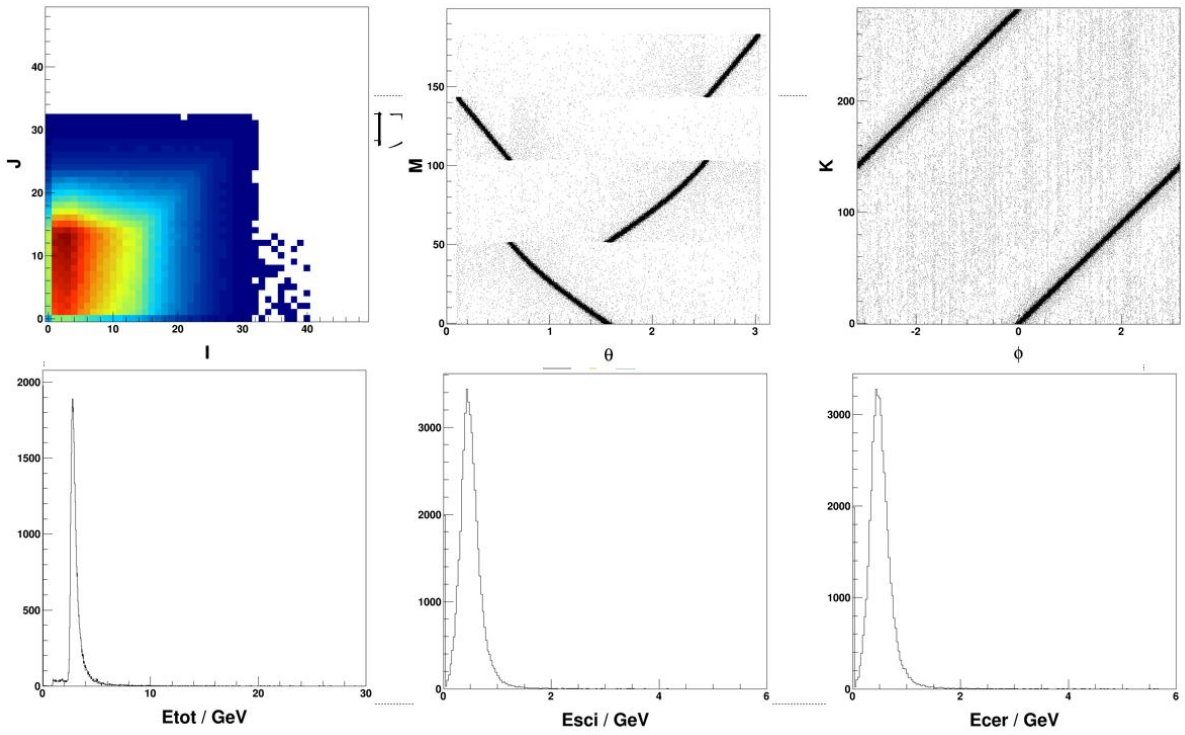

Figure 19: Some distributions for single muon from 1 GeV to 39 GeV. (a) Hit map in tower, I and J are the fiber ID. (b) The tower theta ID M and the theta angle. (c) The tower phi ID K and the phi angle. (d) The total deposited energy in both type of fibers. (e) The deposited energy in the scintillator fibers. (f) The deposited energy in the Cherenkov fibers.

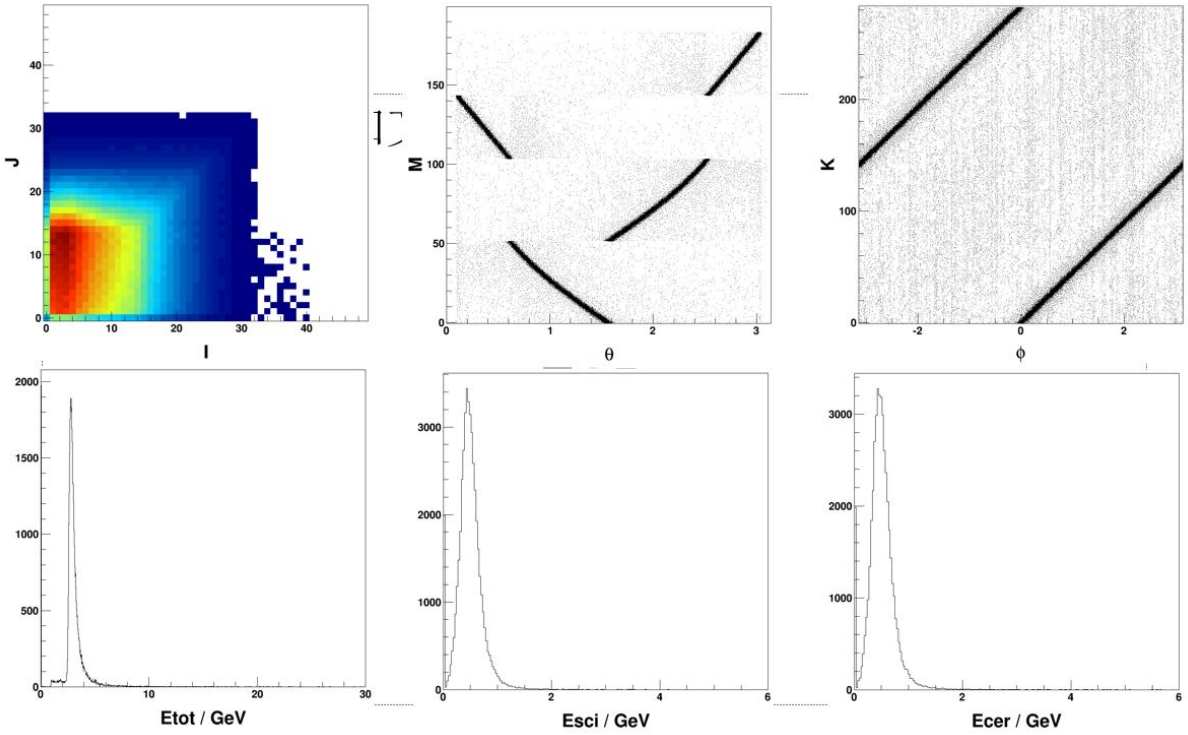

Figure 20: Some distributions for single electron at 20 GeV. (a) Hit map in tower, I and J are the fiber ID. (b) The tower theta ID M and the theta angle. (c) The tower phi ID K and the phi angle. (d) The total deposited energy in both type of fibers. (e) The deposited energy in the scintillator fibers. (f) The deposited energy in the Cherenkov fibers.