Geant4 and Its Applications

Kyungho Kim[1]

[1] Sungkyunkwan University, Korea

khkim4@skku.edu

March 20th , 2026

Contents

What is Geant4?

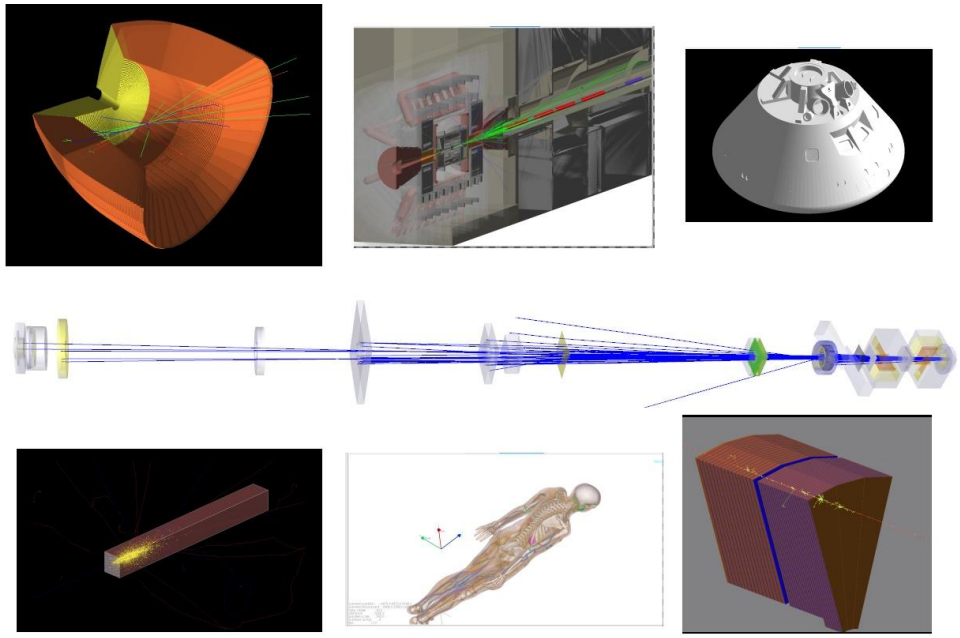
Geant4

Geant4: a toolkit for the simulation of the passage of particles through matter

- General-purpose Monte Carlo simulation tool for elementary particles passing through and interacting with matter.
- Variety of user domains including high energy and nuclear physics, space engineering, medical applications, material science, radiation protection and security.

Geant4 offers most, if not all, of the functionalities required for the simulation of elementary particle and nucleus passing through and interacting with matter.

- Kernel
- Geometry and navigation
- Physics processes
- Scoring
- GUI and Visualization drivers



Modular C++ design allows easy user extensions.

Extensive user guide documents and examples are provided.

Geant4 - Geometry

Richest collection of shapes

- CSG (Constructed Solid Geometry), Boolean operation (of solid), Tessellated solid (with CAD), etc.
- The user can easily extend

Describing a setup as hierarchy or ‘flat’ structure

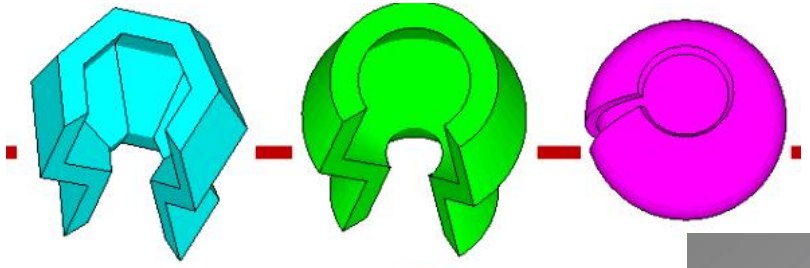
- Describing setups up to billions of volumes
- Tools for creating & checking complex structures

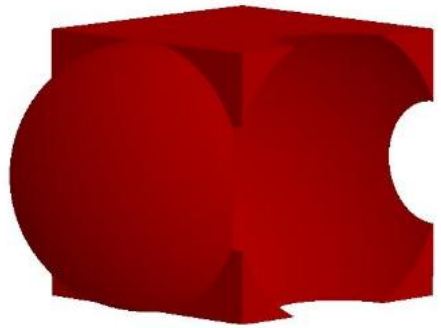
Navigating fast in complex geometry model

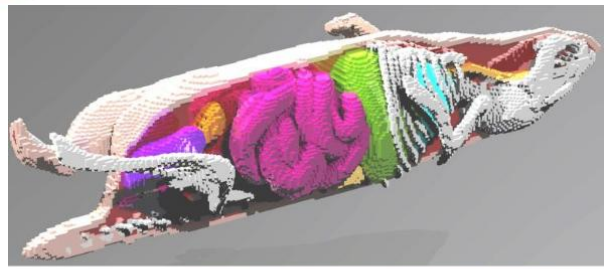
- Automatic optimization

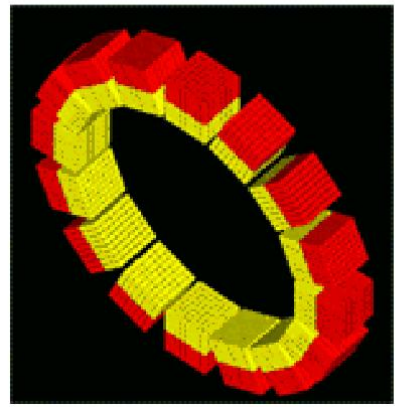
Geometry models can be ‘dynamic’

- Changing the setup at run-time, e.g. “moving objects”

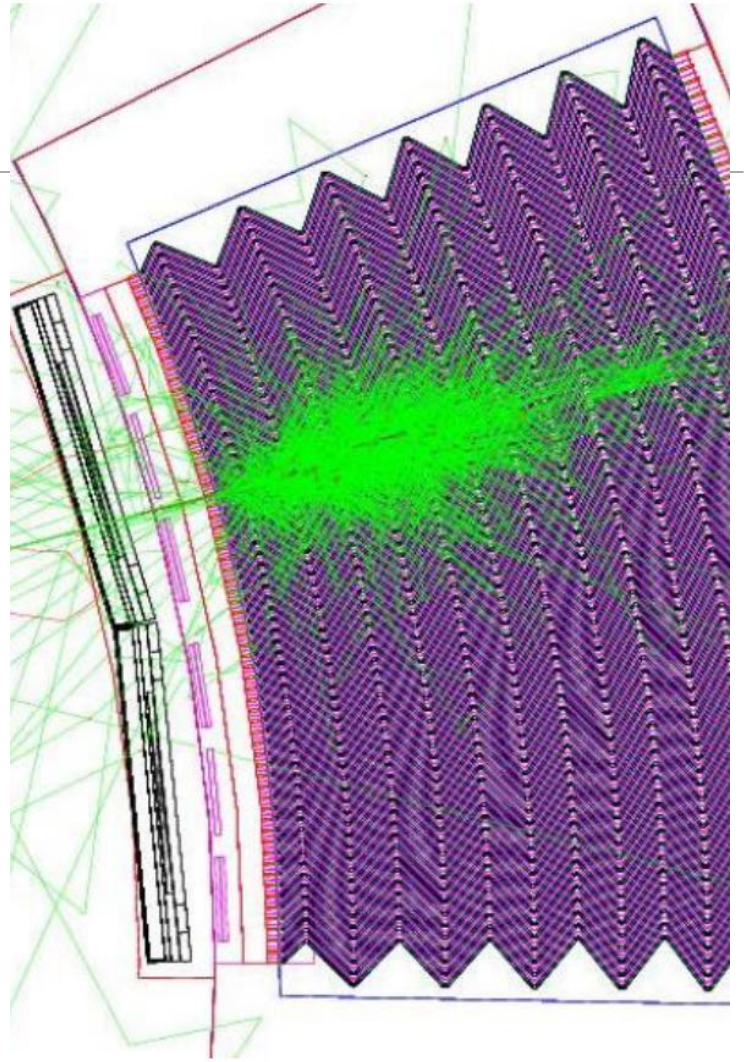








Geant4 – Physics models



Geant4 offers

- Electromagnetic processes
- Hadronic and nuclear processes
- Photon/lepton-hadron processes
- Optical photon processes
- Decay processes
- Shower parameterization
- Event biasing techniques
- And you can plug-in more

Geant4 provides sets of alternative physics models

- Users can freely choose appropriate models according to the type of his/her application.
- For example, some models are more accurate than others at a sacrifice of speed.

How Geant4 Works

Terminology - Run, Event, Track and Step

Run

- Starts with ‘beamOn’.
- Collection of events which share the same detector and physics conditions
- G4RunManager class manages processing a run.
- A run is represented by G4Run class or a user-defined class derived from G4Run.

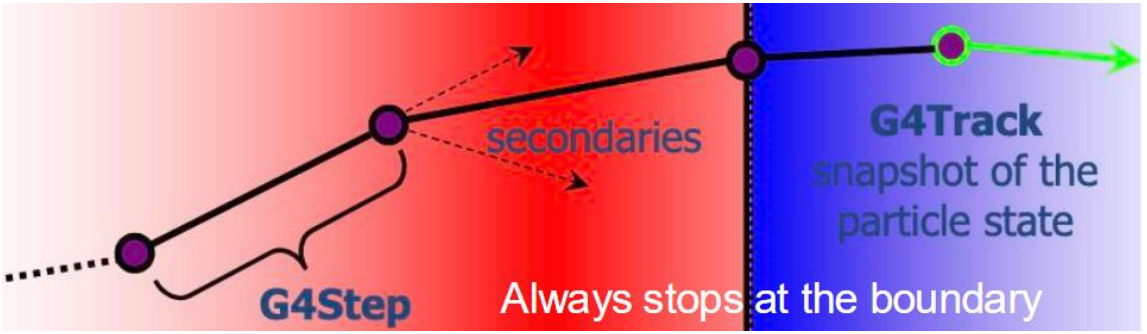
Event

- The basic unit of simulation in Geant4
- At beginning of processing, primary tracks are generated and pushed into a stack.
- Tracks in stack are popped up from the stack one by one. Resulting secondary tracks are pushed into the stack.
- Event is over when stack becomes empty.
- G4EventManager class manages processing an event. G4Event class represents an event.

Terminology - Run, Event, Track and Step

Track: a snapshot of a particle

- Step is a ‘delta’ information to a track. Track is being updated by steps.
- Track is deleted when
  - Going out of the world volume
  - Disappears (decay, inelastic scattering, etc.)
  - Zero kinetic energy and no ‘atRest’ additional process is required
  - User decides to kill it
- No track object persists at the end of event.



Step

- ‘delta’ information of a particle between two points (PreStepPoint, PostStepPoint)
- Because one step know materials of two volumes, boundary processes such as transition radiation or refraction could be simulated.

Detector

Implement Construct() and ConstructSDandField() methods

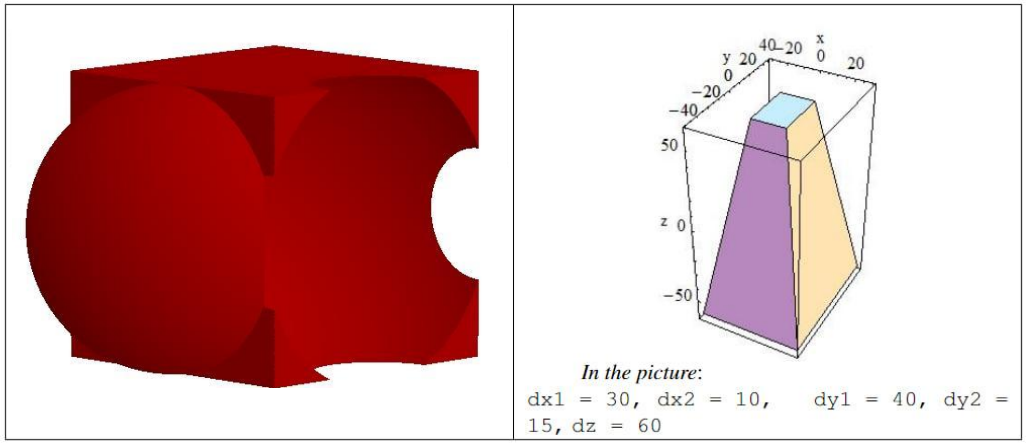
- Construct all necessary materials
- Define shapes/solids
- Define logical volumes
- Place volumes of your detector geometry
- Associate (magnetic) field to geometry (optional)
- Instantiate sensitive detectors / scorers and set them to corresponding logical volumes (optional)
- Define visualization attributes for the detector elements (optional)
- Define regions (optional)

Set your construction class to G4RunManager.

Detector

Volume definition

- Solid: shape, size
- Logical volume: material, sensitivity, user limits, etc.
- Physical volume: position, rotation



Material definition

- Materials (G4Material) in Geant4 (and in the real world) are made of isotopes (G4Isotope), elements (G4Element), molecules and mixtures
- State, temperature, and pressure can be set.
- Geant4 provides a (singleton) class, the G4NistManager, to handle information from the NIST database.

```
G4VPhysicalVolume* DetectorConstruction::Construct(){
  G4NistManager* nist = G4NistManager::Instance();
  G4int Z_nist;
  G4Element* elFe = nist->FindOrBuildElement(Z_nist=26); //Get iron element from NIST
  G4Material* fe = nist->FindOrBuildMaterial("G4_Fe"); //Get iron material from NIST
}
```

Detector – Sensitive Detector

What is a Sensitive Detector?

- A user-defined object attached to a LogicalVolume to **record particle interactions**
- Simulates how a real detector measures signals
- Hit information from sensitive detector is stored in ntuple when event is ended. (EndOfEvent)
- Implemented by inheriting G4VSensitiveDetector

```
// Define SD
auto* sd = new MySensitiveDetector("MySD");

// Register to SDManager
G4SDManager::GetSDMpointer()->AddNewDetector(sd);

// Attach to Logical Volume, usually in ConstructSDandField()
G4SDManager::GetSDMpointer()->SetSensitiveDetector("myLogicalVolume", sd);
```

Generator

Overview

- Defines the initial particles that start each event
- Determines **particle type, energy, position, and direction**
- Implemented by inheriting G4VUserPrimaryGeneratorAction

G4ParticleGun

- Shoots a **single particle** with fixed properties
- Simple and lightweight
- Suitable for basic simulations with a well-defined beam

G4GeneralParticleSource (GPS)

- Supports **energy spectrum, angular distribution, spatial distribution**
- Configurable via macro files without recompiling
- Suitable for realistic or complex beam conditions

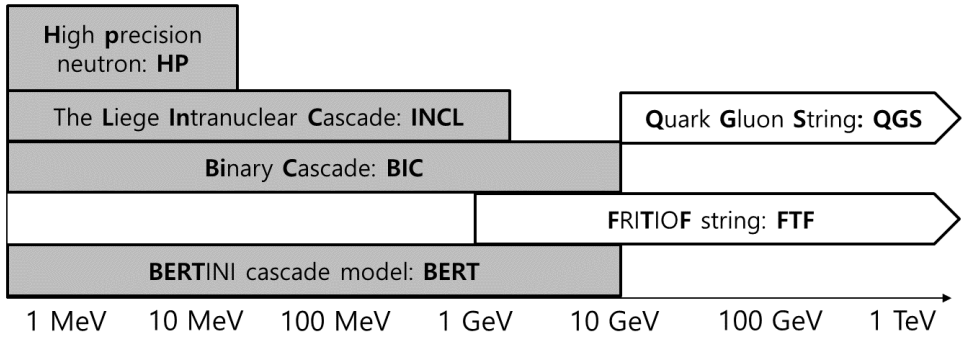
Physics List

The physics list

- is one of the mandatory classes (in every simulation there must one and one only physics list).
- It determines the physics environment to be simulated, i.e.
  - the particles simulated, and
  - the associated processes together with their parameters (cuts, energy ranges for hadronic models, …).

Geant4 provides ready-to-use physics lists, known as *reference physics lists*

- They are located under `/geant4/source/physics_lists/lists`,
  e.g. `FTFP_BERT, QGSP_BERT, FTFP_BERT_HP, QGSP_BIC, QGSP_BIC_HP, QBBC, Shielding, …`
- They are *validated* by the Geant4 community and represent best guesses on the physics needed in a certain domain (HEP, biomedical physics, radiation shielding, …)
- They are adopted (with mild variants) by the vast majority of HEP experiments (ATLAS, CMS, ALICE, LHCb, …)



Application: Heavy ion beam simulation

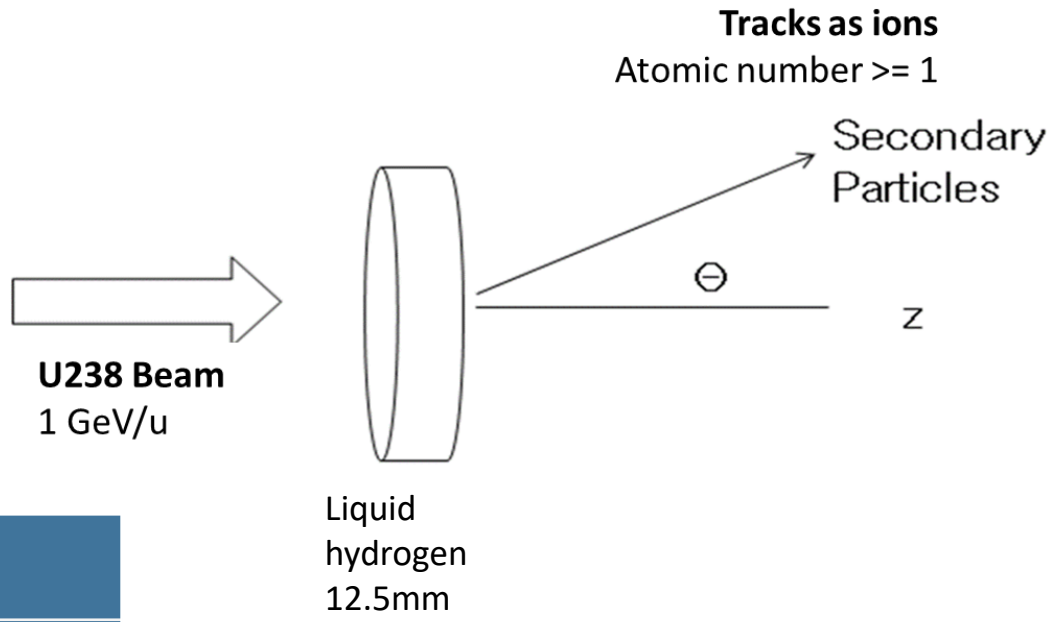
Purpose and Conditions

Purpose of the simulation

- To study secondary beam after beam collision
- To validate the best simulation condition

Conditions of simulation

- Geant4 (version 11.0.2)
- 1 million events per each condition
- On KISTI-5 supercomputer (Nurion)



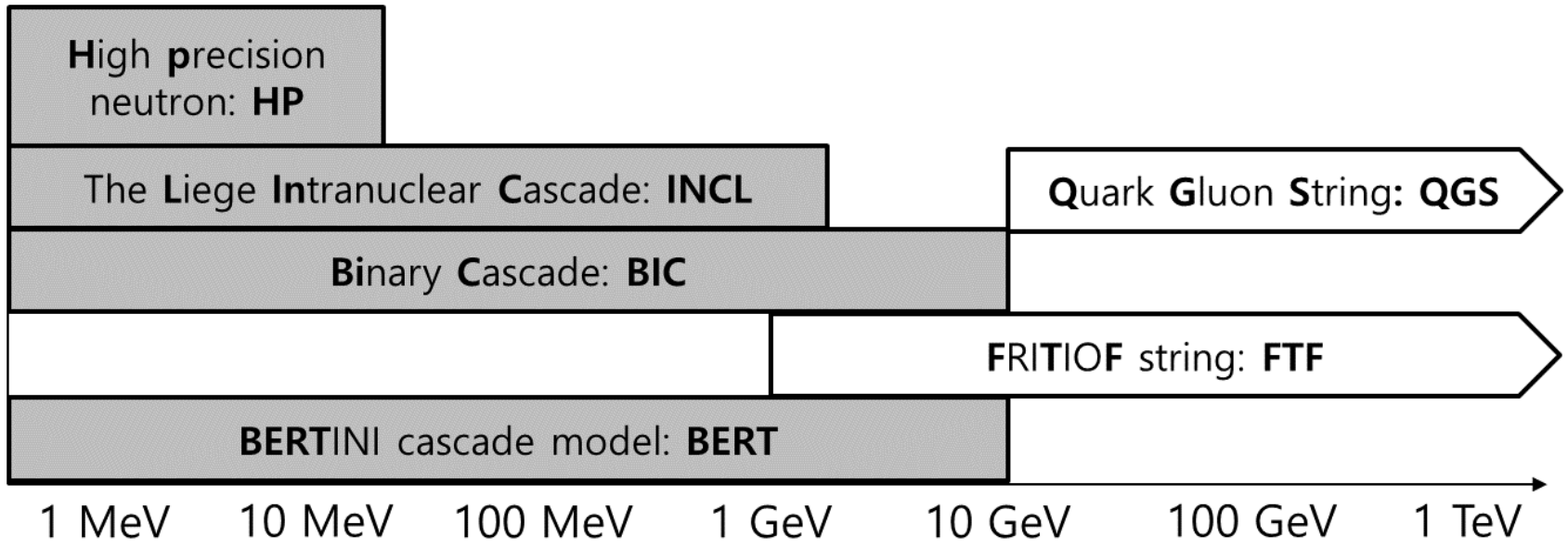
Schematic of simulation

Simulation	Geant4 beam		Target	
	Particle	Energy (MeV/u)	Materials	Thickness (mm)
U → Liquid Hydrogen	U	1000	Liquid Hydrogen	12.5

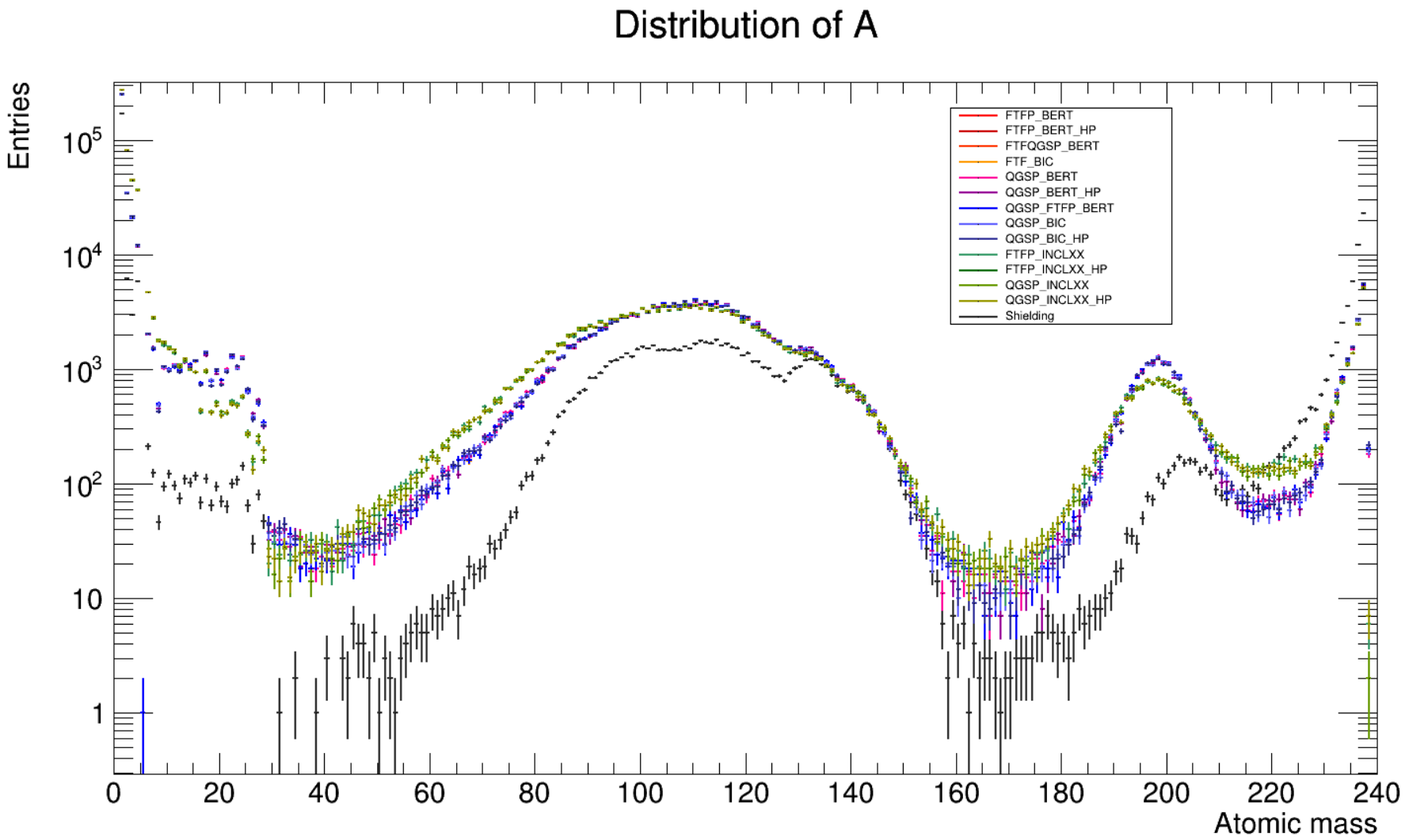
Considered ‘Physics List’

Considered physics list in Geant4

- FTFP_BERT, FTFP_BERT_HP, FTFQGSP_BERT, QGSP_FTFP_BERT
- FTF_BIC, QGSP_BERT, QGSP_BERT_HP, QGSP_BIC
- FTFP_INCLXX, FTFP_INCLXX_HP, QGSP_INCLXX, QGSP_INCLXX_HP
- Shielding



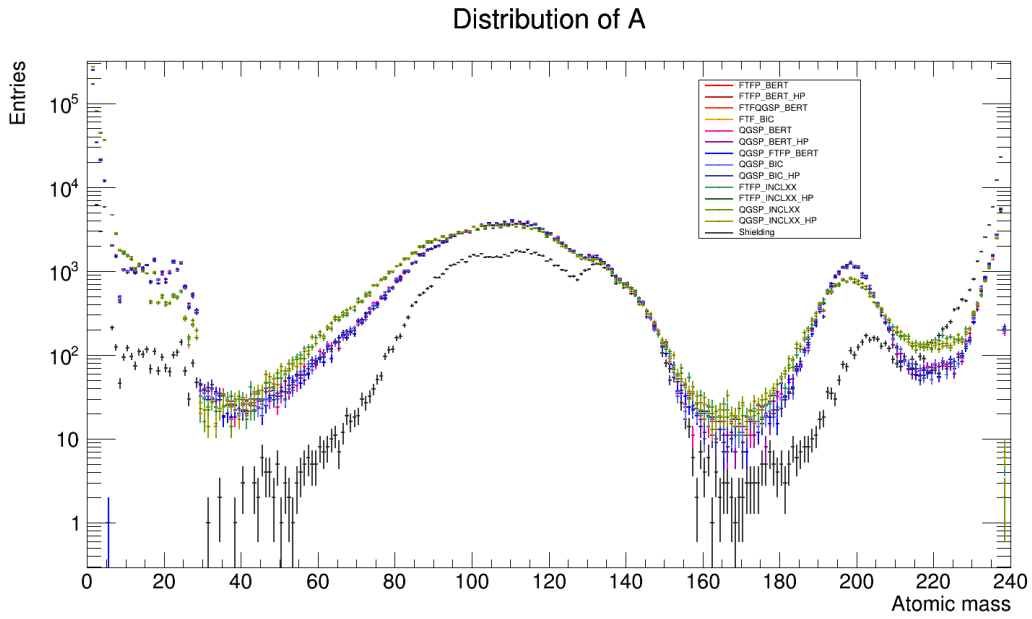
Distribution: Atomic Mass of Secondaries

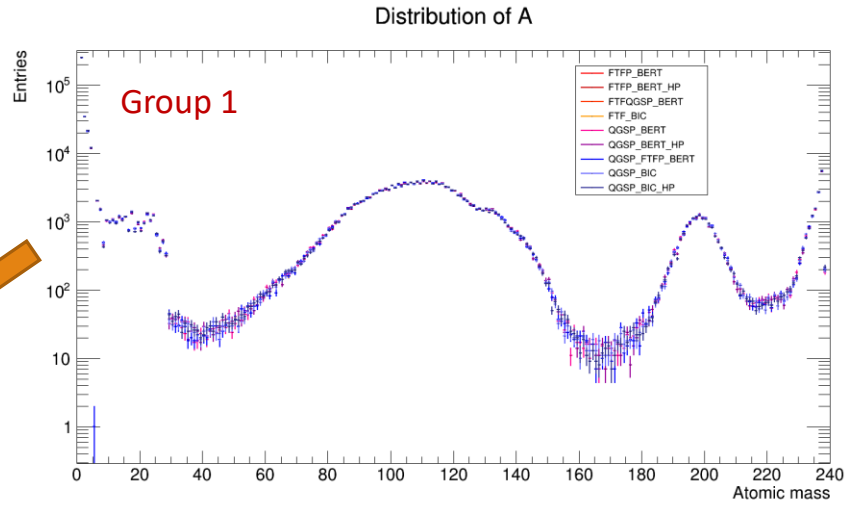


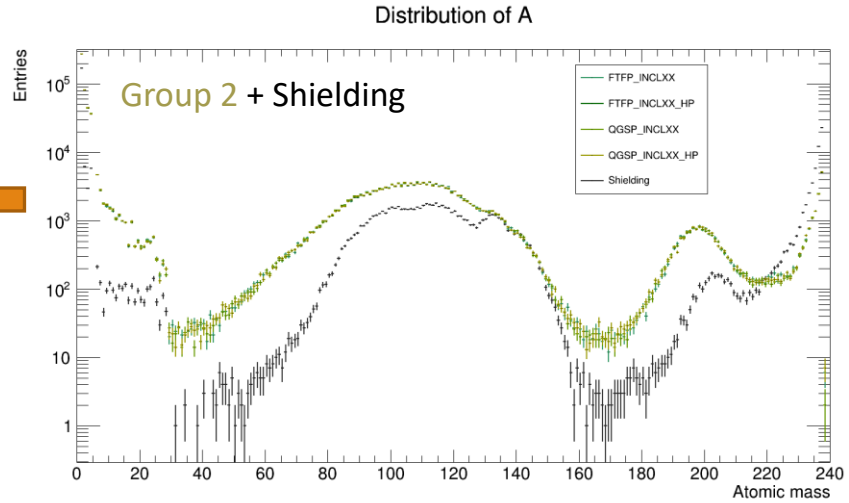
Distribution: Atomic Mass of Secondaries

PhysicsLists are separated to 3 groups

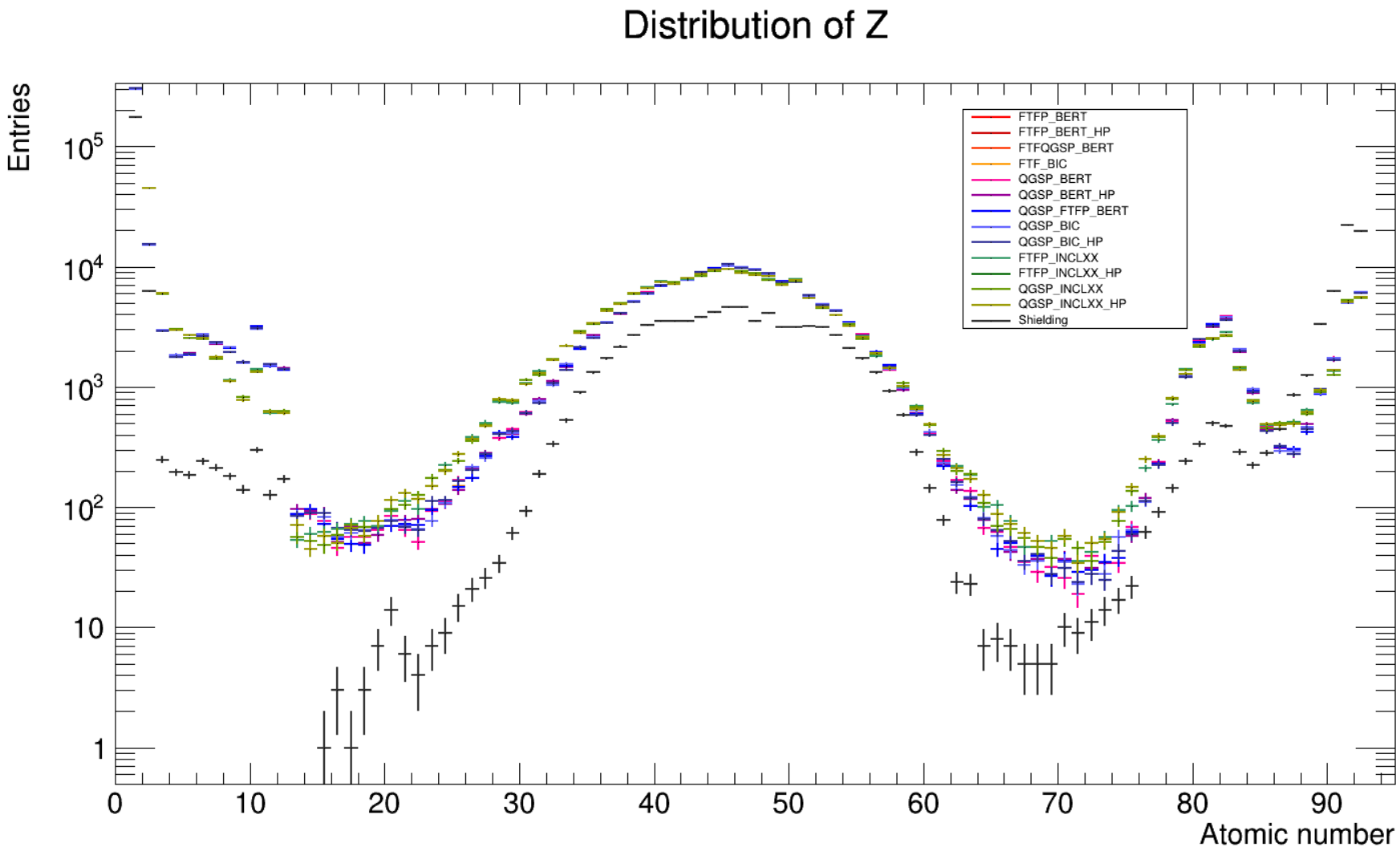
- Group 1: FTFP_BERT, FTFP_BERT_HP, FTFQGSP_BERT, FTF_BIC, QGSP_FTFP_BERT, QGSP_BERT, QGSP_BERT_HP, QGSP_BIC, QGSP_BIC_HP
- Group 2: FTFP_**INCLXX**, FTFP_**INCLXX**_HP, QGSP_**INCLXX**, QGSP_**INCLXX**_HP
- **Shielding**







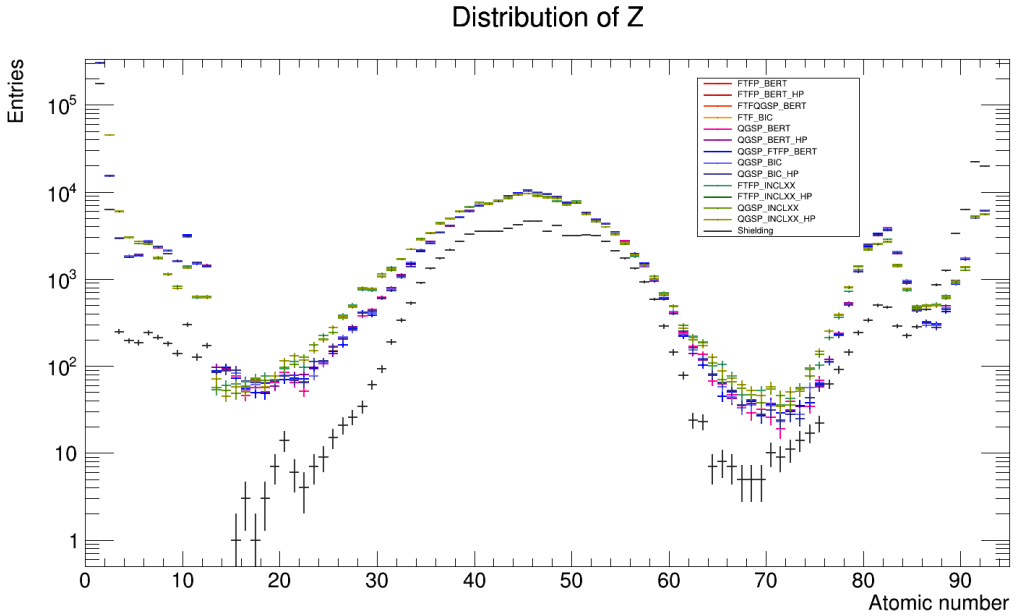
Distribution: Atomic Number of Secondaries

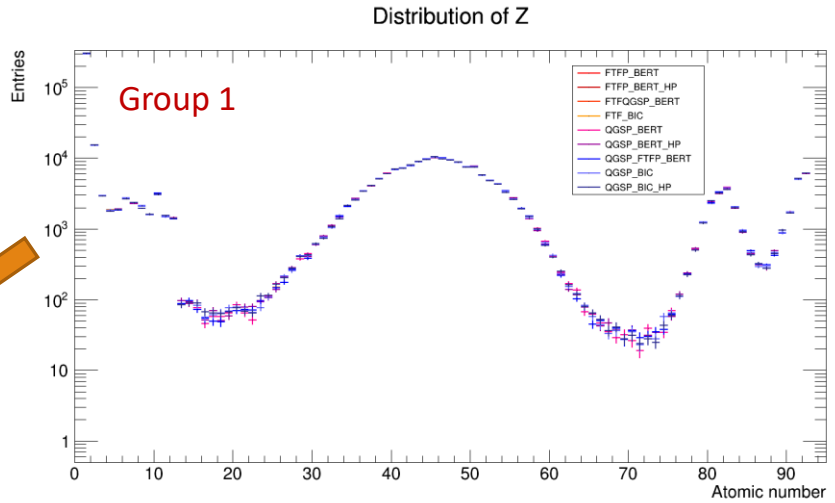


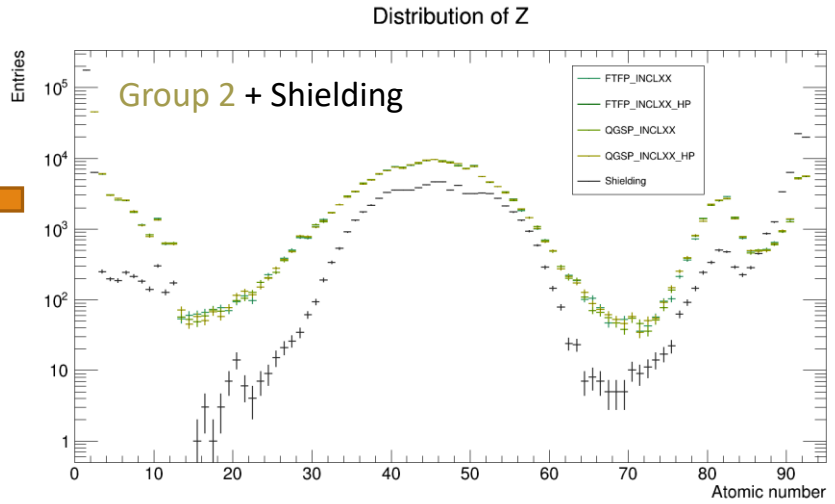
Distribution: Atomic Mass of Secondaries

PhysicsLists are separated to 3 groups

- Group 1: FTFP_BERT, FTFP_BERT_HP, FTFQGSP_BERT, FTF_BIC, QGSP_FTFP_BERT, QGSP_BERT, QGSP_BERT_HP, QGSP_BIC, QGSP_BIC_HP
- Group 2: FTFP_**INCLXX**, FTFP_**INCLXX**_HP, QGSP_**INCLXX**, QGSP_**INCLXX**_HP
- **Shielding**







Finding the Best Physics List

The best physics list for studying heavy-ion beam simulation

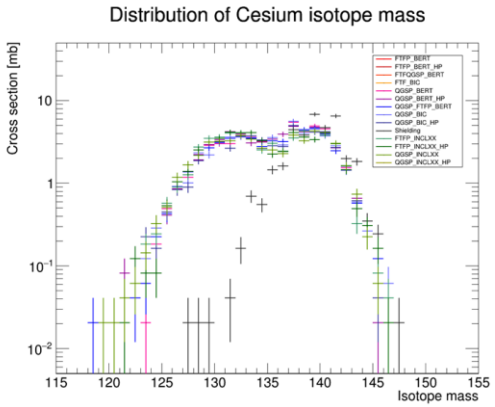
- The validity compared with experiment

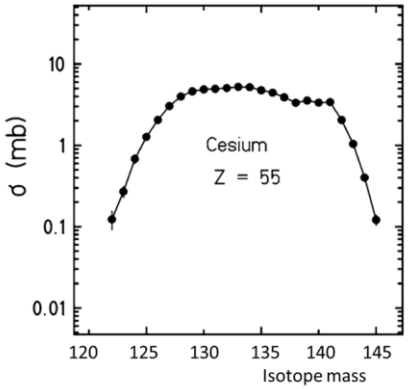
- Cost-effective CPU time

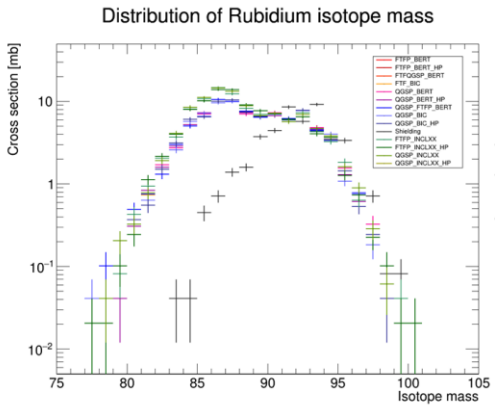
- The amount of the secondary particles

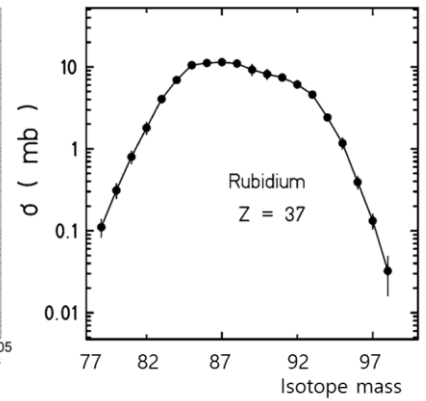
Isotope Distribution

M. Bernas, et al., Nucl. Phys. A 725, 213 (2003).

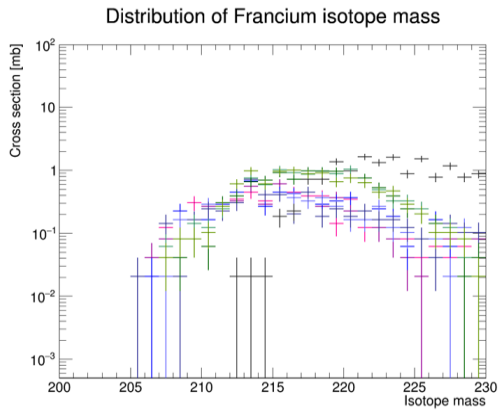


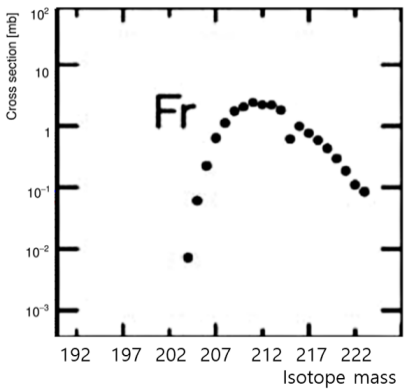






M. Bernas, et al., Nucl. Phys. A 725, 213 (2003).





J. Taieb, et al. Nucl. Phys. A 724, 413 (2003).

Computing Times and Created Secondaries

Computing time

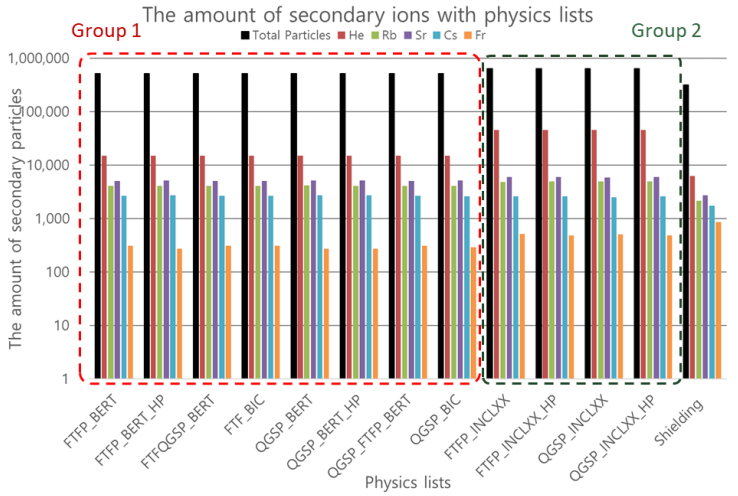
- Excluding ‘Shielding’ for our suitable list (due to long runtime)
- Others looks same.
  - Any physics list (except Shielding) would be OK.

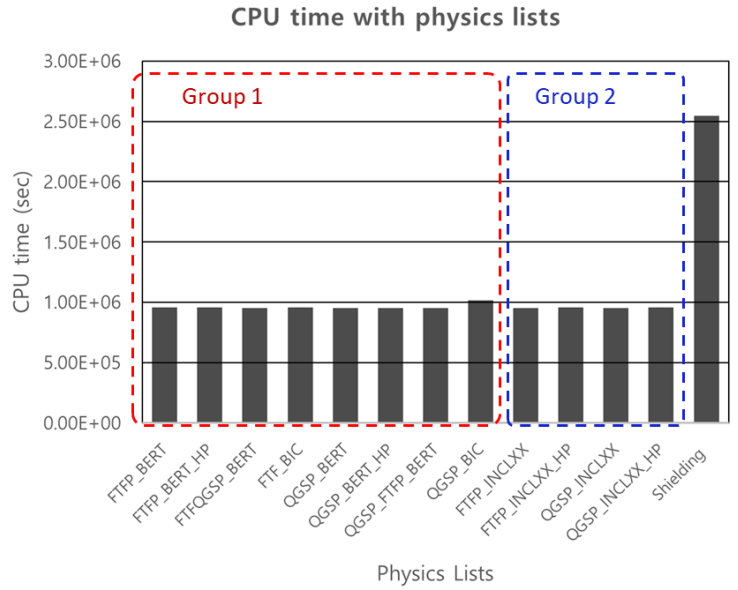
Creation of secondary particles

- Group 2 physics list take more secondary ions.
- More detailed description than physics lists in group 1

What would be the best?

- **FTFP_INCLXX**

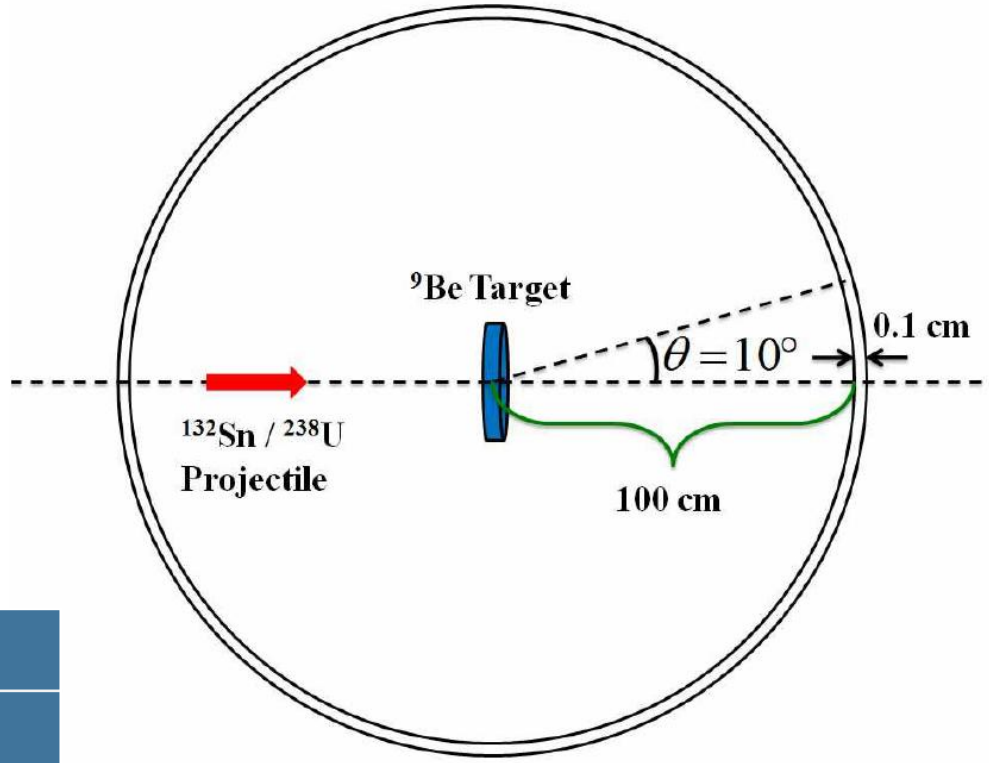




Conditions of Application

Conditions of experiments

- Geant4 version: 11.0.2
- 100,000 events per each condition
- PhysicsList: FTFP_INCLXX
- Ion counting condition
  - Escaped from Be target
  - Travel 100cm from target
  - Forward direction with θ < 10°

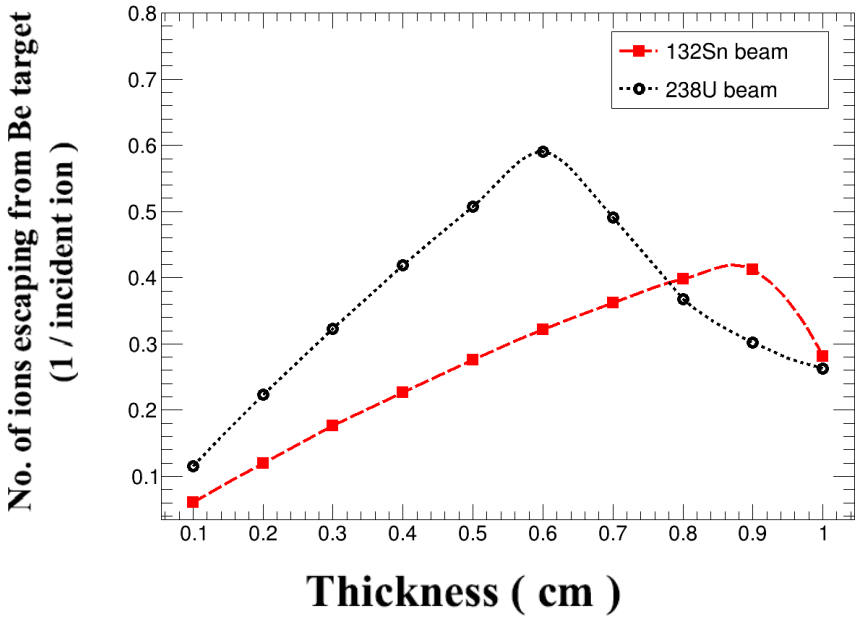


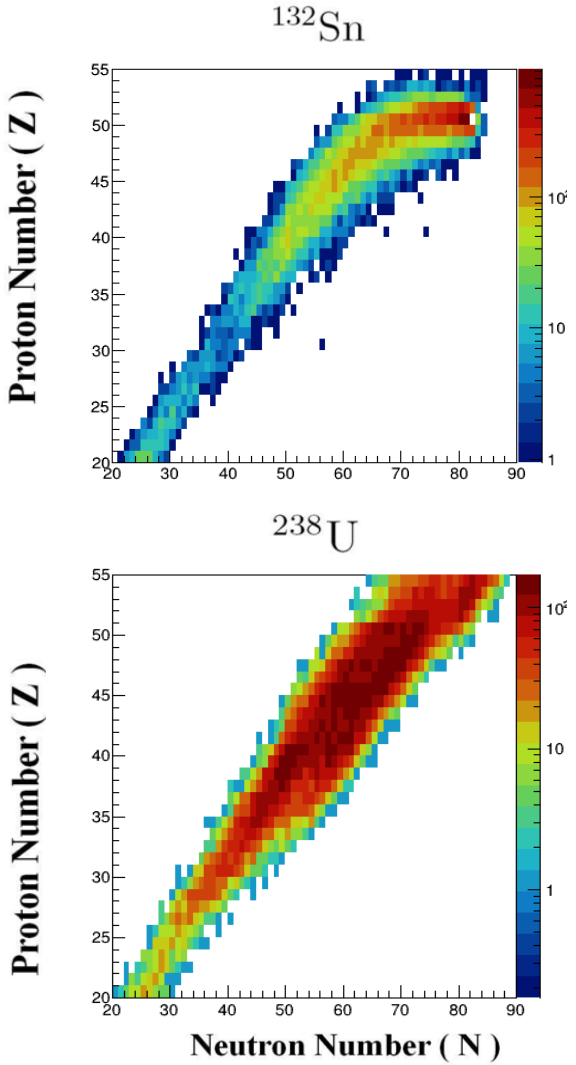
Simulation	Geant4 beam		Target	
	Particle	Energy (MeV/u)	Materials	Thickness (mm)
Sn → Be	Sn132	200	Beryllium	1 ~ 10 mm (1mm step)
U → Be	U238	200		

Result of Simulation

Tendency on distribution

- Thin target: bean penetrate the target
- Thick target: secondary ions cannot escape from target
- What is the best target thickness for given condition?
  - 0.6cm for Uranium and 0.9cm for Tin



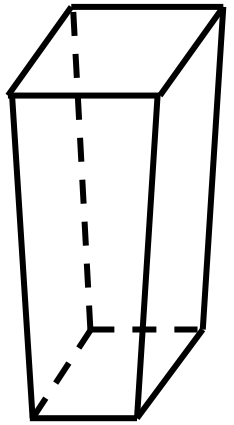


Application: ECL calorimeter

Construction of ECL Calorimeter

Beam control: on macro (not in PrimaryGeneratorAction)

- /gun/direction 0 0 1 (z-axis)
- /gun/direction 0 1 1 (yz-plane)



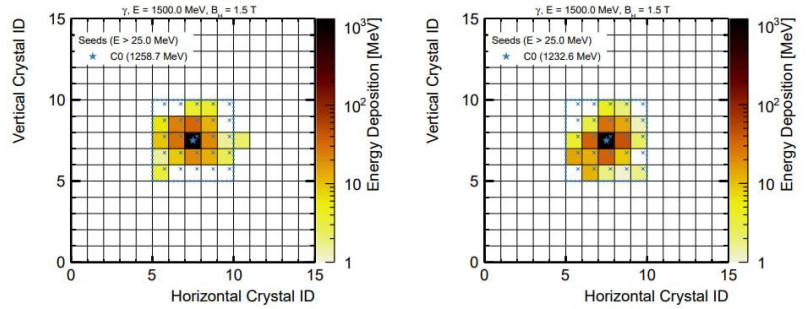
Geometry setup

1st step: Crystal array structure on 2D plane

- Simplified simulation with center-oriented beam
- Example: Design of the ECL Software for Belle II (BELLE2-NOTE-TE-2016-001)
  - 15 x 15 array with 6 x 6 x 30 cm^3 crystal box with 1.5T homogeneous B-field

2nd step: Belle II barrel – shaped array structure

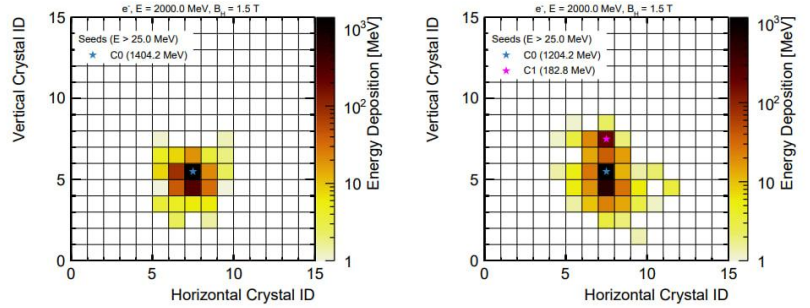
- Use G4Trap to make center-oriented and cylindrical crystal array structure
- One trapezoid crystal takes 2.5° of angle -> 144 crystals



(a) Photon event #12 (One CR, one Seed, no track).

(b) Photon event #49 (One CR, one Seed, no track).

FIG. 1: 1.5 GeV photon. In addition to the seed cell a 5 × 5 area around the seed is marked. The text in brackets is detailed in Section 3.



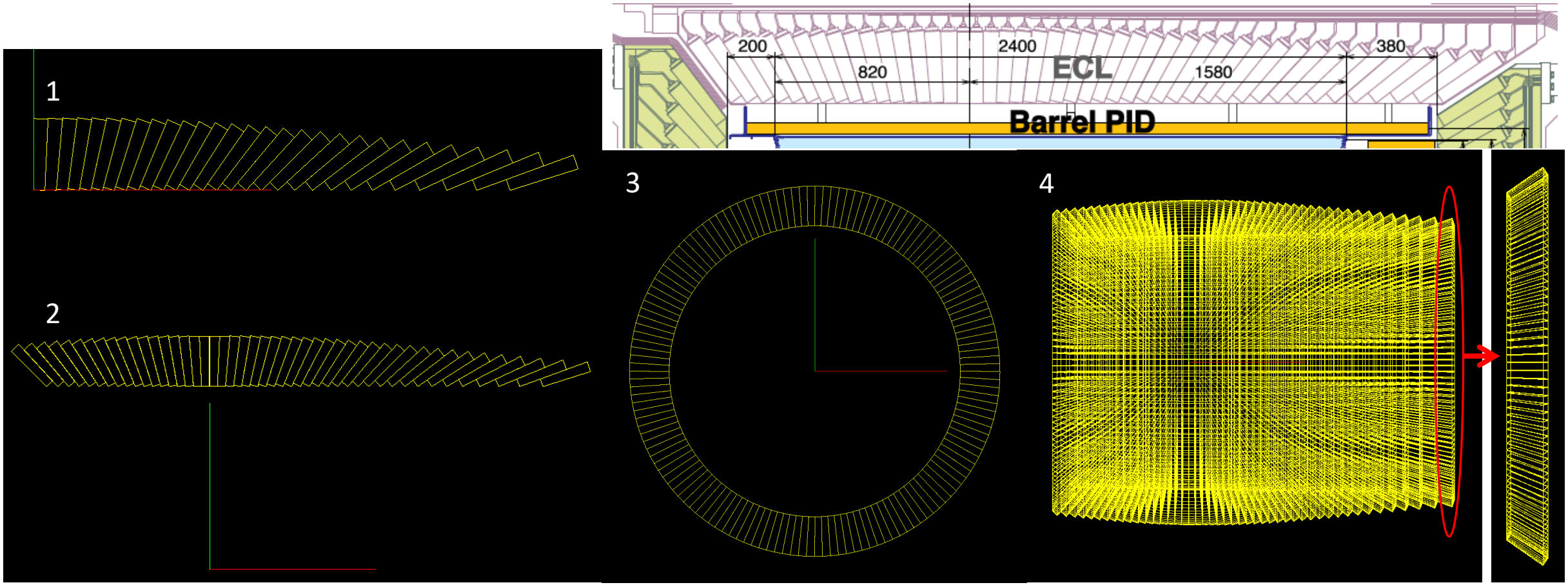
(a) e^- event #28 (One CR, one seed, one track).

(b) $e^-\gamma$ event #2, $E_\gamma = 0.25$ GeV (One CR, two seeds, one track).

FIG. 2: 2.0 GeV e^-. The text in brackets is detailed in Section 3.

Construction of ECL Calorimeter

Previous result of barrel construction (2011)



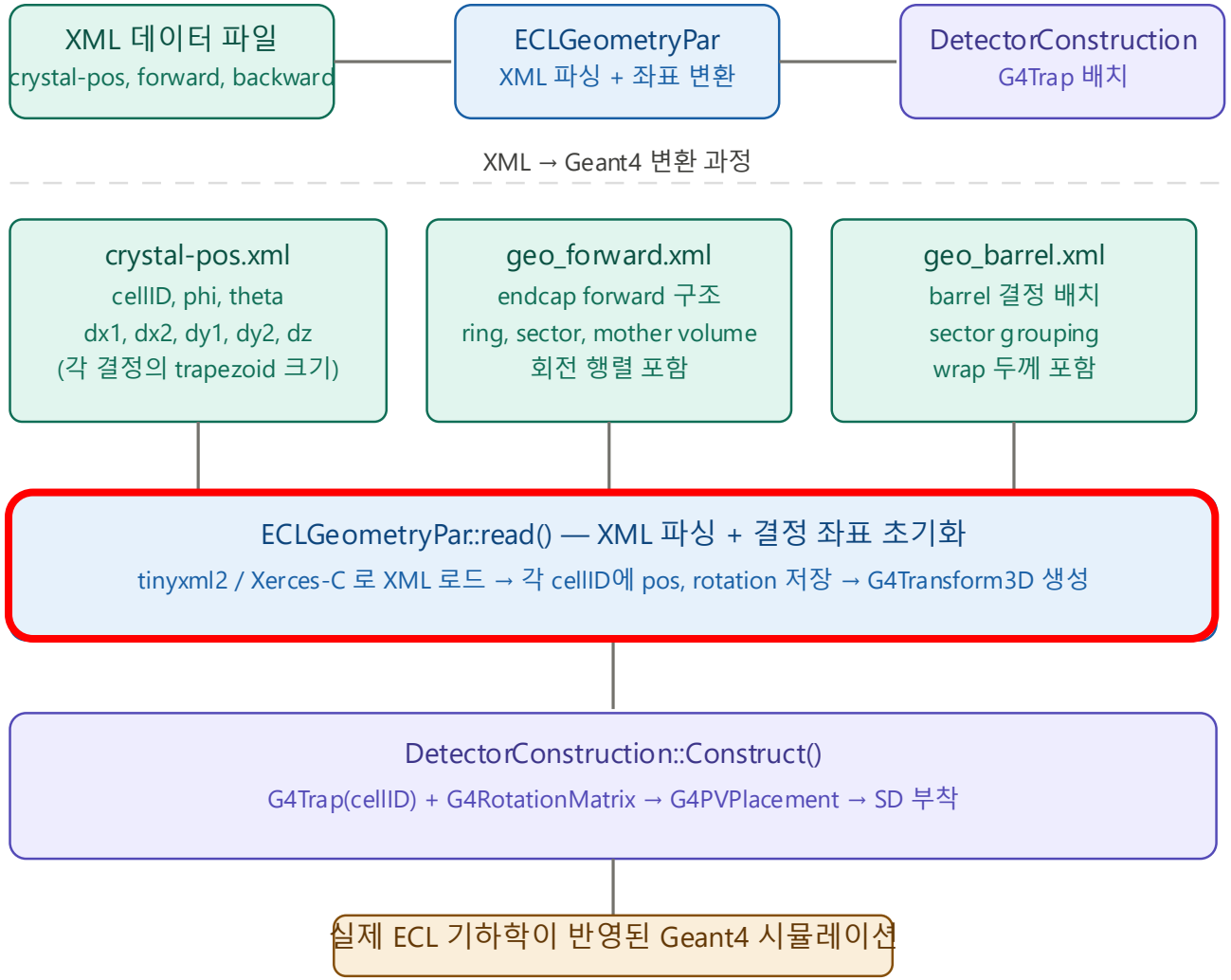
Construction of ECL Calorimeter

Future plan

- Constructing actual Belle II ECL calorimeter with XML database
- Structure of XML database

```
<CrystalInfo>
  <Crystal cellid="1">
    <Position x="-0.0124" y="124.6" z="194.0"/>   <!-- cm -->
    <Direction nx="0.0" ny="0.9486" nz="0.3162"/>
    <Size dx1="5.5" dx2="6.2" dy1="5.5" dy2="6.2" dz="30.0"/>  <!-- mm -->
  </Crystal>
  <!-- ... 8736개 -->
</CrystalInfo>
```

Construction of ECL Calorimeter



Summary

Geant4 is a general-purpose Monte Carlo simulation tool for elementary particles passing through and interacting with matter.

- Geant4 simulation requires 1) Detector geometry, 2) Beam generator, and 3) Physics list.
- In Geant4 simulation, run-event-track-step terminology describes the simulation.

Research of heavy ion beam simulation with Geant4 validates most probable physics lists and corresponding secondary particle generation.

For the simulation of ECL calorimeter with simplified geometry is enough for beginning. As describing full ECL geometry of 8736 crystals, XML parsing would be required.

Thank you for listening!

Backups

Single-element material

If a material is made of a *single element* (Fe, Pb, …) its definition is straightforward.

`DetectorConstruction.cc` – Create pure iron material

```
#include "G4Material.hh"
#include "G4SystemOfUnits.hh"

G4VPhysicalVolume* DetectorConstruction::Construct(){
  G4int atomic_number = 26;               // iron atomic number (z)
  G4double density = 7.87*g/cm3;          // iron density g/cm3
  G4double atomic_weight = 55.845*g/mole; // iron atomic weight 55.845
  G4Material* pureIron = new G4Material("pureIron", atomic_number,
atomic_weight, density);

}
```

Molecules

A molecule is defined by (> 1) elements and its composition is specified with the `AddElement` method.

`DetectorConstruction.cc` - Create water material as H2O molecule

```
#include "G4Material.hh"
#include "G4SystemOfUnits.hh"
#include "G4Element.hh"
G4VPhysicalVolume* DetectorConstruction::Construct(){

  G4int ncomp = 2;                      // water components
  G4double water_density = 1.0*g/cm3; // water density
  G4Material* H2O = new G4Material("Water", water_density,ncomp); // water material
  G4double a = 1.01*g/mole;             // Hydrogen(element): A
  G4int z = 1;                          // Hydrogen(element): Z
  G4Element* elH = new G4Element(“Hydrogen”,"H",z,a); // Hydrogen(element)
  a = 16.00*g/mole;
  z = 8;
  G4Element* elO = new G4Element("Oxygen","O",z,a);
  G4int nAtoms;
  H2O->AddElement(elH, nAtoms=2);   // add element by number of atoms
  H2O->AddElement(elO, nAtoms=1);   // add element by number of atoms
}
```

Mixtures

Similarly to combining elements in molecules, it is possible to combine materials and elements in mixtures.

`DetectorConstruction.cc` - Construct simplified air mixture

```
#include "G4Material.hh"
#include "G4SystemOfUnits.hh"
#include "G4Element.hh"
G4VPhysicalVolume* DetectorConstruction::Construct(){

  // Simplified Air, mass fraction: 70% Nitrogen, 30% Oxigen
  //
  G4int z;
  G4int ncomponents;
  G4double a = 14.01*g/mole;
  G4Element* elN = new G4Element(name="Nitrogen",symbol="N",z= 7.,a);
  G4double a = 16.00*g/mole;
  G4Element* elO = new G4Element(name="Oxygen",symbol="O",z= 8.,a);
  density = 1.290*mg/cm3;
  G4Material* Air = new G4Material(name="Air",density,ncomponents=2);
  Air->AddElement(elN, 70.0*perCent); //add element by frac mass
  Air->AddElement(elO, 30.0*perCent); //add element by frac mass
}
```

`DetectorConstruction.cc` - Construct aerogel mixture

```
#include "G4Material.hh"
#include "G4SystemOfUnits.hh"
#include "G4Element.hh"
G4VPhysicalVolume* DetectorConstruction::Construct(){

  // Simplified Aereogel material, frac mass: 62.5% SiO2, 37.4% H20
  // 0.1% C
  G4Element* elC = ... ; // define carbon element
  G4Material* H2O = ... ; // define water molecule (previously done)
  G4Material* SiO2 = ... ; // define SiO2 molecule (left as exercise)

  G4double density = 0.20*g/cm3;
  G4int ncomp = 3;
  G4double fracMass;
  G4Material* Aerog = new G4Material("Aerogel", density, ncomp); //
Aereogel material
  // Note that we are combining elements with materials
  Aerog->AddMaterial(SiO2, fracMass = 62.5*perCent);  // first component
  Aerog->AddMaterial(H2O, fracMass = 37.4*perCent);   // second component
  Aerog->AddElement(elC, fracMass = 0.1*perCent);     // third component
}
```

Isotopes

By default any G4Element is treated according to its natural isotopic abundance regardless of the atomic weight specified

`DetectorConstruction.cc` - Retrieve number of iron isotopes

```
G4VPhysicalVolume* DetectorConstruction::Construct(){

  G4int atomic_number = 26;
  G4double density = 7.87*g/cm3;
  G4double atomic_weight = 55.845*g/mole;
  G4Material* pureIron =
     new G4Material("pureIron", atomic_number, atomic_weight, density);
  auto element = pureIron->GetElement(0);
  G4cout<<pureIron->GetName()<<" # isotopes: "<<element-
>GetNumberOfIsotopes()<<G4endl;
}
```

`Bash` - execution

```
pureIron # isotopes: 4
```

→ Iron has 4 naturally abundant isotopes: ^{54}Fe, ^{56}Fe, ^{57}Fe, ^{58}Fe

Isotopes

It is possible to create an element with non natural isotopic abundance assigning to it a list of `G4Isotopes`.

`DetectorConstruction.cc` - Create nuclear fuel (UF6 - uranium hexafluoride)

```
G4VPhysicalVolume* DetectorConstruction::Construct(){

  G4int z_iso, a_iso; // Create uranium 235 and 238
  G4Isotope* u235 = new G4Isotope("U235",z_iso=92,a_iso=235.,235.044*g/mole);
  G4Isotope* u238 = new G4Isotope("U238",z_iso=92,a_iso=238.,238.051*g/mole);
  G4int u_comp;       // Create enriched uranium
  G4double u_abundance;
  G4Element* enrichedU = new G4Element("enrichedU","eU",u_comp=2);
  enrichedU->AddIsotope(u235,u_abundance=5.0*perCent);  // add isotope to enrichedU
  enrichedU->AddIsotope(u238,u_abundance=95.0*perCent); // add isotope to enrichedU
  G4Element* elF = new G4Element("Fluorine","F",9,18.998*g/mole);

  // Create nuclear fuel (UF6)
  //
  G4Material* fuel = new G4Material("NuclearFuel",5.09*g/cm3,ncomp=2, // mandatory arguments
  kStateGas,640*kelvin,1.5e7*pascal);      // optional arguments
  // set to STP if not specified
  fuel->AddElement(elF,6);
  fuel->AddElement(enrichedU,1);
}
```

The NIST material database

The NIST material database includes

- most of the materials useful in simulations (biomedical, space-science, …)
- all elements with natural isotopic abundance
- > 3000 isotopes

Using materials from the NIST database guarantees the best parameters for

- density
- isotopic composition of elements
- element composition of materials
- mean ionization potential
- chemical bonds

NIST: National Institute of Standards and Technology

`Example` – Print list of NIST materials

```
/material/nist/listMaterials # UI command
```

`Bash` – execution

```
======================================================
### Simple Materials from the NIST Data Base ###
======================================================
Z    Name       density(g/cm^3)  I(eV)
======================================================
1    G4_H       8.3748e-05       19.2
2    G4_He      0.000166322      41.8
3    G4_Li      0.534            40
4    G4_Be      1.848            63.7
5    G4_B       2.37             76
6    G4_C       2                81
7    G4_N       0.0011652        82
8    G4_O       0.00133151       95
9    G4_F       0.00158029       115
10   G4_Ne      0.000838505      137
11   G4_Na      0.971            149
12   G4_Mg      1.74             156
13   G4_Al      2.699            166
```

Using the NIST material database

Geant4 provides a (singleton) class, the G4NistManager, to handle information from the NIST database.

`DetectorConstruction.cc` - Get iron element and material information directly from NIST database

```
G4VPhysicalVolume* DetectorConstruction::Construct(){

  G4NistManager* nist = G4NistManager::Instance();
  G4cout<<"Natural uranium mass: "<<nist->GetAtomicMassAmu(92)<<" amu"<<'\n'<<
  "Hydrogen Nb of isotopes: "<<nist->GetNumberOfNistIsotopes(1)<<'\n'<<
  "Iron material density: "<<nist->GetNominalDensity(26)/(g/cm3)<<" g/cm3"<<G4endl;

  // Or print information per material/elements using
  nist->PrintElement("Al"); // equivalent to UI command /material/nist/printElement Al
}
```

`Bash` - execution

```
Natural uranium mass: 238.029 amu
Hydrogen Nb of isotopes: 6 ← ???
Iron material density: 7.874 g/cm3
```

Using the NIST material database

Geant4 provides a (singleton) class, the G4NistManager, to handle information from the NIST database.

DetectorConstruction.cc (B1) - Get various materials from NIST

```
// Get nist material manager
 G4NistManager* nist = G4NistManager::Instance();

 // Envelope parameters
 G4Material* env_mat = nist->FindOrBuildMaterial("G4_WATER");
 //...
 // World
 G4Material* world_mat = nist->FindOrBuildMaterial("G4_AIR");
 //...
 // Shape 1
 G4Material* shape1_mat = nist->FindOrBuildMaterial("G4_A-150_TISSUE");
 //...
 // Shape 2
 G4Material* shape2_mat = nist->FindOrBuildMaterial("G4_BONE_COMPACT_ICRU");
```

Naming conventions

Physics lists names are given as **String(s)_Cascade_Neutron_EM**

- Hadronic options
  - **QGS** Quark Gluon String model (> ~15-20 GeV)
  - **FTF** FRITIOF String model (> ~3 GeV)
  - **BIC** Binary Cascade model (< ~6 GeV)
  - **BERT** Bertini Cascade model (< ~6 GeV)
  - **P** G4Precompound model used for de-excitation
  - **HP** High Precision neutron model (< ~20 MeV)
- Electromagnetic options
  - (**No suffix** or **EM0**)
  - **EMV**
  - **EMY**
  - **EMZ**

Reference physics lists

Recommended PL for HEP applications
(calorimeters, collider physics, …)

The **FTFP_BERT PL** is the recommended one for high-energy physics simulations. It adopts the models:

- **FTF**: (Fritiof) hadronic string model for hadron inelastic process > 3 GeV
- **P**: Precompound model for nucleus de-excitation
- **BERT**: (Bertini) intra-nuclear cascade model for the hadron inelastic process < 6 GeV
- Plus: neutron radiative capture, nuclear capture at rest for negatively charged hadrons, elastic scattering for hadrons, lepto-nuclear and gamma-nuclear processes and standard em physics (*Compton, pair-production, photoelectric effect*, …)

NOTE: The energy range transition between cascade and string models is typically chosen with thin target data

NOTE: Hadronic showers are (often) quite sensitive to it *e.g.*

ATLAS changed the energy range between FTFP and BERT to [9,12] GeV while

CMS changed the energy range between FTFP and BERT only for pions to [3,12] GeV

Reference physics lists

Other relevant reference physics lists for hadronic physics are

- The **FTFP_BERT_HP** PL: same as FTFP_BERT but using the high-precision neutron treatment for low energy neutrons (< 20 MeV)

  Recommended PL for shielding and space application

- The **QGSP_BERT** PL: same as FTFP_BERT but uses the QGS (Quark Gluon String) model for the hadron inelastic process with an energy transition range with FTF of [12,25] GeV

  Alternative for FTFP_BERT at high energies (> 25 GeV)

  NOTE: Below 12 GeV QGSP_BERT and FTFP_BERT are identical

- The **FTFP_INCLXX** PL: uses the INCLXX model for the inelastic process of protons, neutrons and charged pions. The INCLXX and the FTF models intersect in the energy range [15,20] GeV

  Alternative for FTFP_BERT at low energies (< 20 GeV)

- The **QGSP_BIC** PL: uses both FTF and QGS models for high-energy interactions and both BERT and BIC (Binary Cascade) models for low-energy interactions, as

  Recommended for medical applications
  (also QGSP_BIC_HP)

  - Protons, neutrons: BIC < 6 GeV, FTFP [3, 25] GeV and QGSP > 12 GeV
  - Pions and kaons: BERT < 6 GeV, FTFP [3, 25] GeV and QGSP > 12 GeV

Reference physics lists

Relevant physics list options for electromagnetic physics are

- PL+(**No suffix** or **_EM0**): uses the G4EmStandardPhysics constructor, implements default EM physics

  ← Recommended EM option for HEP applications (sampling calorimeters, …)

- PL+**_EMV**: uses the G4EmStandardPhysics_option1 constructor, implements fast but less precise EM physics due to simple multiple scattering step limitation

  ← Recommended EM option for HEP applications (homogeneous calorimeters, …)

- PL+_EMY: uses the G4EmStandardPhysics_option3 constructor, implements Urban MSC model for all particles (medical, space science, precise)

  ← Recommended em option for medical applications (proton/ion therapy, …)

- PL+_EMZ: uses the G4EmStandardPhysics_option4 constructor, implements the most accurate em physics, e.g. an error-free stepping for $e^{\pm}$

  ← The most accurate em physics description. Can be used as a reference for all the cases above.

Introduction

The default Geant4 simulation flow sets up the **geometry**, the **physics list** and the **primary generator**, transports particles and…

⇒ *no part for permanent data storing*

Bash - execution

```
$ source /path-to/geant4_11.1.2-install/bin/geant4.sh
$ cmake -DGeant4_DIR=/path-to/geant4_11.1.2-install/lib/Geant4-11.1.2/ /path-
to/geant4-11.1.2/examples/basic/B1/ .
$ make
$ ./exampleB1 run1.mac

**********************************************************
 Geant4 version Name: geant4-11-01-patch-02 [MT]
 << in Multi-threaded mode >>

$ ls
CMakeCache.txt  Makefile   exampleB1   exampleB1.out  run1.mac      vis.mac
CMakeFiles      cmake_install.cmake    exampleB1.in   init_vis.mac  run2.mac
```

g4analysis

- In Geant4 the user is responsible for storing the variables of interest.
- The Geant4 analysis category g4analysis (geant4/source/analysis/) provides a unique interface:
  - to write histograms and n-tuples (the so-called primitive types), and
  - to specify the format (ROOT, XML, CSV, HDF5)
- Good choice to support the large and heterogeneous user community that adopts different tools (Python, ROOT in HEP, …) to run the analysis a posterior

g4analysis

- **g4analysis** is available in Geant4 since December 2011
  - It is an active area of development with new features added at every release
  - It ensures input/output (I/O) thread-safe capability in multi-threaded simulations (take a second to appreciate how cool is that…)

- **NOTE**: Users can still link their applications against external libraries to handle I/O
  (typically, ROOT::TTree for small applications and Event Data Models for large applications)

- Users access the g4analysis tools via the G4AnalysisManager

G4AnalysisManager

The G4AnalysisManager

- is a singleton (users must access it with static method ::Instance())
- handles output file(s) creation
- owns and handles histos and n-tuples

It provides

- uniform user interface to different formats (ROOT, XML, CSV, HDF5),
  i.e. the user is not required to know how to handle any of them,
- with high-level memory management and access to low-level objects (e.g. n-tuples columns)

and is fully integrated in the Geant4 framework
(e.g. it is accessible via user commands and uses G4-units)

NOTE: using hdf5 format requires HDF5 libraries installation and Geant4 build
with -DGEANT4_USE_HDF5=ON

Using G4AnalysisManager

The G4AnalysisManager usage consists of three steps

1. **Create the manager, book n-tuples/histos** and open a file, to be done at the begin of each run (`RunAction::BeginOfRunAction()`)

NOTE: since `Geant4-11.0` the file format is chosen via the extension (`.root, .xml, .csv, .hdf5`)

`RunAction.cc`

```
#include "G4AnalysisManager.hh"
void RunAction::BeginOfRunAction(const G4Run* run) {
  auto analysisManager = G4AnalysisManager::Instance(); //get the manager
  std::string runnumber = std::to_string( run->GetRunID() );
  G4String fileName = "Run" + runnumber + ".root"; //select root format
  //G4String fileName = "Run" + runnumber + ".xml"; //or select xml format
  //G4String fileName = "Run" + runnumber + ".csv"; //or select csv format
  //G4String fileName = "Run" + runnumber + ".hdf5";//or select hdf5 format -> requires hdf5
installation
  analysisManager->OpenFile(fileName);
  //Create histogram(s)
  analysisManager->CreateH1("Edep","Energy deposit",100,0.*MeV,10*GeV); //h1 - energy
deposited
  // Create ntuple(s)
  analysisManager->CreateNtuple("Ntuple", "Ntuple");    // name, information
  analysisManager->CreateNtupleDColumn("Energy");       // variable to save, D: double
  //analysisManager->CreateNtupleIColumn(“trkID");      // int column
  //analysisManager->CreateNtupleSColumn(“name");       // string column
  analysisManager->FinishNtuple();
}
```

Using G4AnalysisManager

The G4AnalysisManager usage consists of three steps

2. **Fill values in histos and n-tuples**, likely at the end of each event (`EventAction::EndOfEventAction()`)

EventAction.cc

```
#include "G4AnalysisManager.hh"
void EventAction::EndOfEventAction(const G4Event*){

  auto analysisManager = G4AnalysisManager::Instance();

  //Fill histograms
  //
  analysisManager->FillH1(0, Edep); //Fill value as MeV
  //Fill ntuples
  //
  analysisManager->FillNtupleDColumn(0, Edep);
  //analysisManager->FillNtupleIColumn(1, track->GetTrackID());
  //analysisManager->FillNtupleSColumn(2, track->GetDefinition()->GetParticleName());

  analysisManager->AddNtupleRow();
}
```

Using G4AnalysisManager

The G4AnalysisManager usage consists of three steps

3. Write and close the file, at the end of each event (`RunAction::EndOfRunAction()`)

RunAction.cc

```
#include "G4AnalysisManager.hh"
void RunAction::EndOfRunAction(const G4Run*) {
  auto analysisManager = G4AnalysisManager::Instance();

  //Write and close file
  //
  analysisManager->Write();
  analysisManager->CloseFile();
}
```

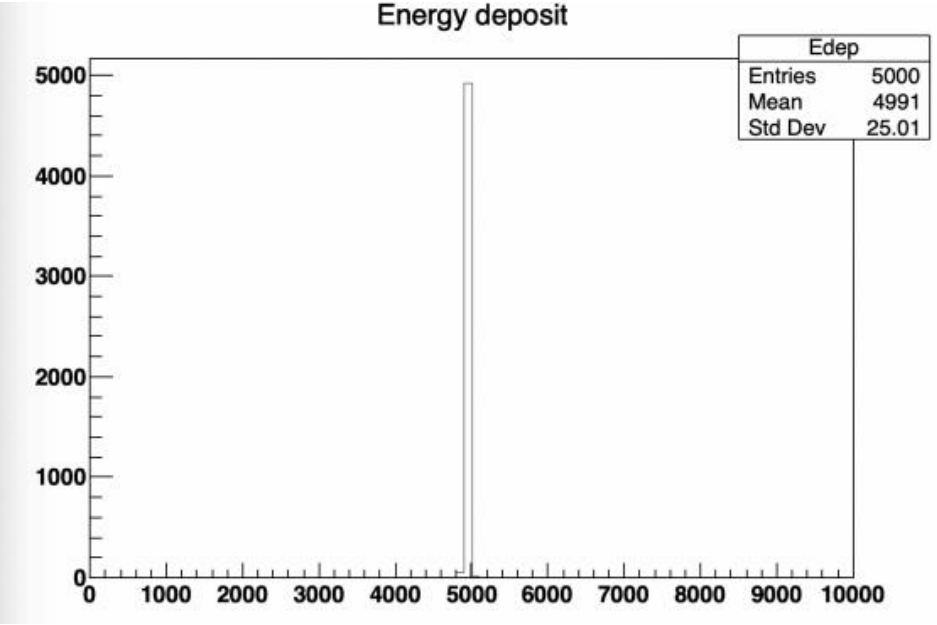
NOTE: performing the **step**s in the suggested classes/methods is not mandatory
but **guarantees correct execution in multi-threaded applications**

⇒ Refer basic example B4b, especially RunData and related code.

Output example - **ROOT**

Bash – after execution

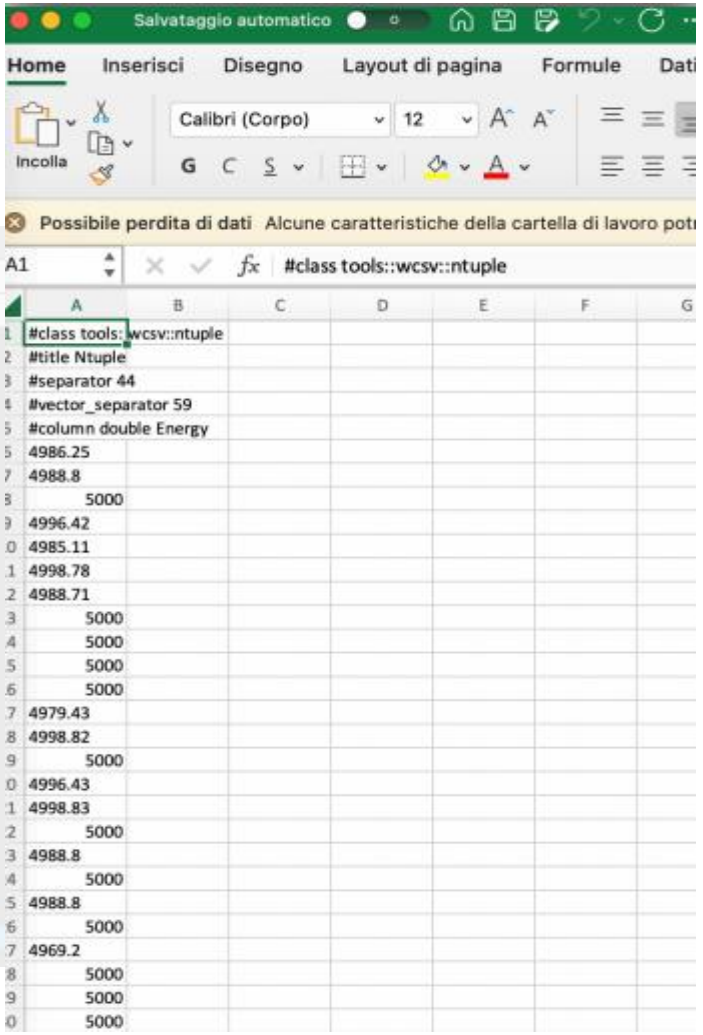
```
$ root -l Run0.root
root [0]
Attaching file Run0.root as _file0...
(TFile *) 0x1258cc990
root [1] .ls
TFile**         Run0.root
 TFile*         Run0.root
 KEY: TTree     Ntuple;1    Ntuple
 KEY: TH1D      Edep;1      Energy deposit
root [2] Edep->Draw(“histo")
root [3] Ntuple->Scan()
************************
*    Row *    Energy.En  *
************************
*      0 *    4986.2465
*      1 *    4988.8028
*      2 *         5000
*      3 *    4996.4155
```



Output example - XML & CSV

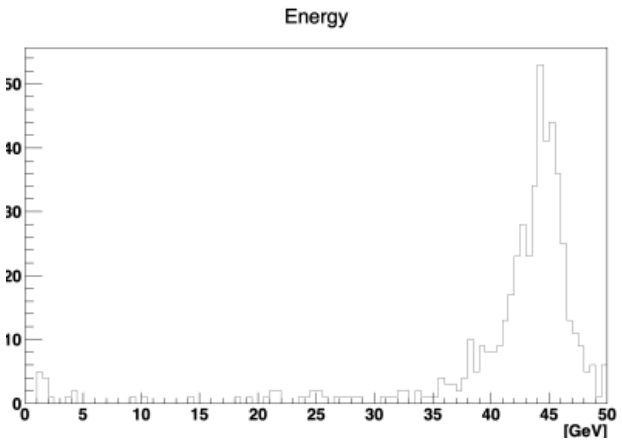
Bash – after execution

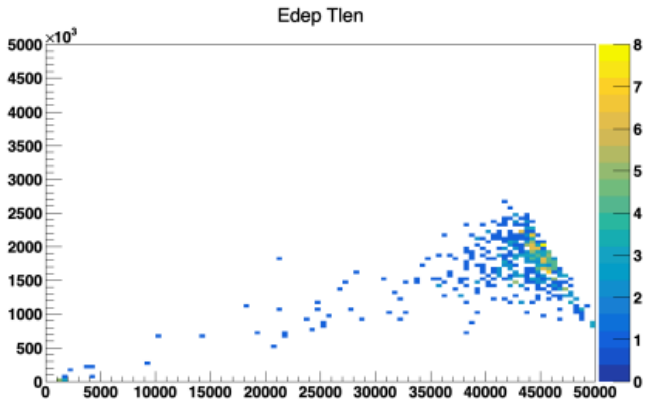
```
$ open Run0_nt_Ntuple.csv
$ open Run0.xml
```

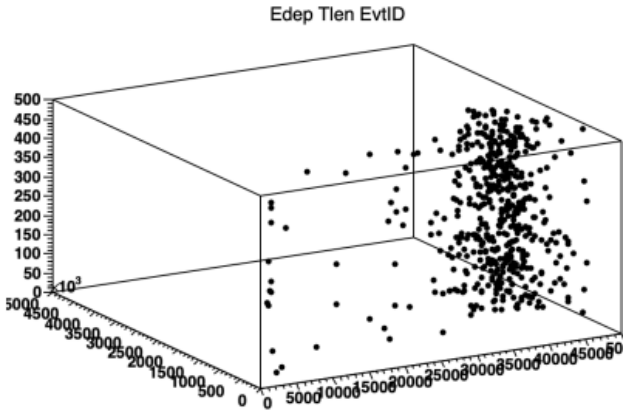


More on histograms

- It is possible to create 1D (CreateH1()), 2D (CreateH2()), 3D (CreateH3()) histograms as well as 1D (CreateP1()) and 2D (CreateP2()) profile histograms.
- **IDENTIFIERS**: each histo ID is automatically generated when the histo is created and its value returned by creating function (histo names are not related to IDs and cannot be used for filling).
  - Default start value is 0, it can be changed with `G4AnalysisManager::SetFirstHistoId(G4int)`
  - IDs for H1, H2, H3, P1 and P2 are defined independently.







More on histograms

It is possible to access histogram objects via the `G4AnalysisManager`.

`RunAction.cc` - accessing histogram(s) mean and rms

```
void RunAction::EndOfRunAction(const G4Run*
/*run*/) {
  auto analysisManager =
G4AnalysisManager::Instance();
  G4cout << G4endl << " ----> print histograms
statistic ";

  if(isMaster) {
    G4cout << "for the entire run " << G4endl
<< G4endl;
  }

 else {
    G4cout << "for the local thread " << G4endl
<< G4endl;
  }

  G4cout << " Edep : mean = "
<< G4BestUnit(analysisManager->GetH1(0)->mean(),
"Energy") << " rms = "
<< G4BestUnit(analysisManager->GetH1(0)->rms(),
"Energy") << G4endl;
}
```

Terminal - Geant4 execution on 2 threads

```
**********************************************
Geant4 version Name: geant4-11-01 [MT] (9-
December-2022)
 << in Multi-threaded mode >>
 Copyright : Geant4 Collaboration
 References : NIM A 506 (2003), 250-303
 : IEEE-TNS 53 (2006), 270-278
 : NIM A 835 (2016), 186-225
 WWW : http://geant4.org/
**********************************************

// some output
G4WT0 > ----> print histograms statistic for
the local thread
G4WT0 > Edep : mean = 41.117 MeV rms = 9.2968
MeV
G4WT1 > ----> print histograms statistic for
the local thread
G4WT1 > Edep : mean = 41.906 MeV rms = 7.36116
MeV
----> print histograms statistic for the
entire run
Edep : mean = 41.5036 MeV rms = 8.41347 MeV
```

More on histograms

Additional histogram properties can be defined via the G4AnalysisManager

- Unit: if defined, filled values are automatically converted to it
  `analysisManager->CreateH1("Tlen","Tracks length",100,0.*km,5.*km,"km")`
- Function: if defined, the function is automatically executed on the filled value (can be log, log10, exp)
  `analysisManager->CreateH1("Tlen","Tracks length",100,0.*km,5.*km,"km","exp")`
  (if both unit and function are defined, unit is applied first)
- Binning scheme: default is linear, possible to set it to logarithm (lin, log)
  `analysisManager->CreateH1("Tlen","Tracks length”,100,0.1*km,5.*km,"none","none","log")`
- Alternative binning scheme: alternative constructor available to set a non-equidistant binning scheme using a vector of bin edges

```
G4int CreateH1(const G4String& name, const G4String& title,
               const std::vector<G4double>& edges,
               const G4String& unitName = "none",
               const G4String& fcnName = "none");
```

More on ntuples

- **IDENTIFIERS**: ntuple and ntuple column IDs are automatically generated and returned by `G4AnalysisManager->CreateNtuple()` and `G4AnalysisManager->CreateNtupleXColumn()`
- IDs must be used to fill columns if more than one ntuple is created.

```
void EventAction::EndOfEventAction(const G4Event* event){

  auto analysisManager = G4AnalysisManager::Instance();

  //Fill ntuple 0 and 1
  //
  analysisManager->FillNtupleDColumn(0, 0, Edep); //ntuple ID, column ID, value
  analysisManager->FillNtupleDColumn(0, 1, TrackL);
  analysisManager->AddNtupleRow(0);

  analysisManager->FillNtupleDColumn(1, 0, Edep/2.);
  analysisManager->FillNtupleDColumn(1, 1, TrackL/2.);
  analysisManager->AddNtupleRow(1);
}
```

- The following types can be saved in columns: `int (I), float (F), double (D), string (S), std::vector of int (I), float (F), and double (D).`

g4analysis - UI commands

Several UI commands are available to interact with the G4AnalysisManager in your macro card.

- General options and handling histos

```
/analysis/verbose 1                              # Set verbose level
/analysis/h1/create Edep Energy 100 0. 100. GeV # create H1, ID=0
/analysis/h1/set 0 100 0. 50. GeV                # set H1 nbins, min, max
```

(similar commands available for H2, H3, P1 and P2)

- Handling output file

```
/analysis/setFileName name                       # set output file name
/analysis/setHistoDirName histos                 # store hist in directory
/analysis/setNtupleDirName ntuples               # store ntuple in directory
```

g4analysis - UI commands

Several UI commands are available to interact with the G4AnalysisManager

in your macro card.

- Histograms control (similar commands available for H2, H3, P1 and P2

```
/analysis/h1/setAscii id true|false # activate printing h1 on ASCII file
/analysis/h1/setTitle id title      # Set title for the 1D histogram
/analysis/h1/setXaxis id title      # Set x-axis, y-axis title for the 1D histogram
/analysis/h1/setYaxis id title
```

- Batch graphics (similar commands available for H2, H3, P1 and P2)

```
/analysis/h1/setPlotting id true|false      # Automatic plotting of h1
/analysis/h1/setPlottingToAll true|false
```

also available as plain C++ `analysisManager->SetH1Plotting(id, true)`,

the selected objects will be automatically plotted in a single (.ps) file with page size fixed to A4 format.