ISAPP School 2024 · KIT / Bad Liebenzell

GEANT4 Simulations for Rare Event Searches

Holger Kluck · HEPHY · holger.kluck@oeaw.ac.at

2/3: Geometry, Primary Particle Generation, Particle Tracking & Data Storage

Geometry

DetectorConstruction; Solid Volumes; Logical Volumes; Material Definition; Physical Placement

DetectorConstruction

```cpp
namespace G4minWE {

class DetectorConstruction : public G4VUserDetectorConstruction {
public:

    //Let C++ define default constructor and destrcutor
    DetectorConstruction() = default;
    ~DetectorConstruction() override = default;

    //This method is needed; it will assemble the actual
    //geometry of the setup to be simulated
    G4VPhysicalVolume* Construct() override;
};
}
```

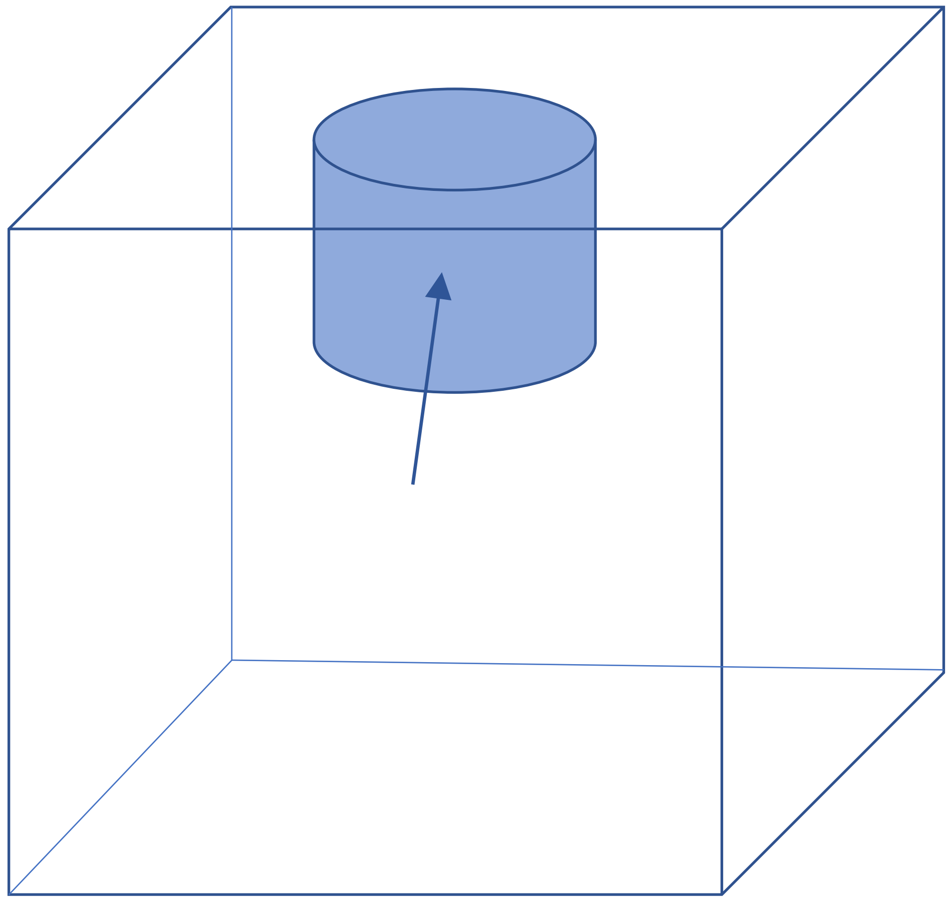
- In Geant4, the user has to **derive** a **concrete subclass** from the **abstract base class** `G4VUserDetectorConstruction` and implement the **method** `Construct()`
- Geant4 calls this method to get a `G4VPhysicalVolume*` that represent the geometry of the experiment one wants to simulate

DetectorConstruction

```
/*-Setup run manager and user classes------------------------------
auto* runMgr = new G4RunManager;
//Set the detector construction
runMgr->SetUserInitialization(new G4minWE::DetectorConstruction);
//Set the physics list
runMgr->SetUserInitialization(new Shielding);

/*-Initialise visualisation manager--------------------------------
G4VisManager* visMgr = new G4VisExecutive;
visMgr->Initialize();
```

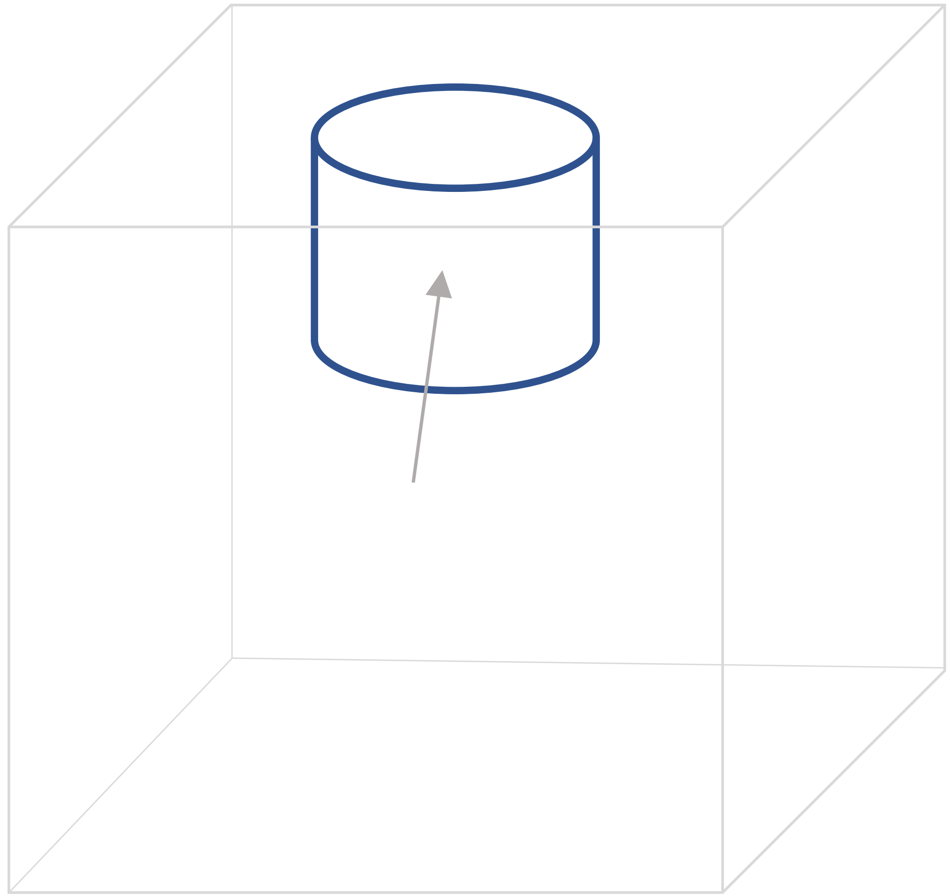
- In the main function, a pointer to an instance of `DetectorConstriction` has to be passed to Geant4's `G4RunManager` via its `SetUserInitialization` method

Aspects of geometry



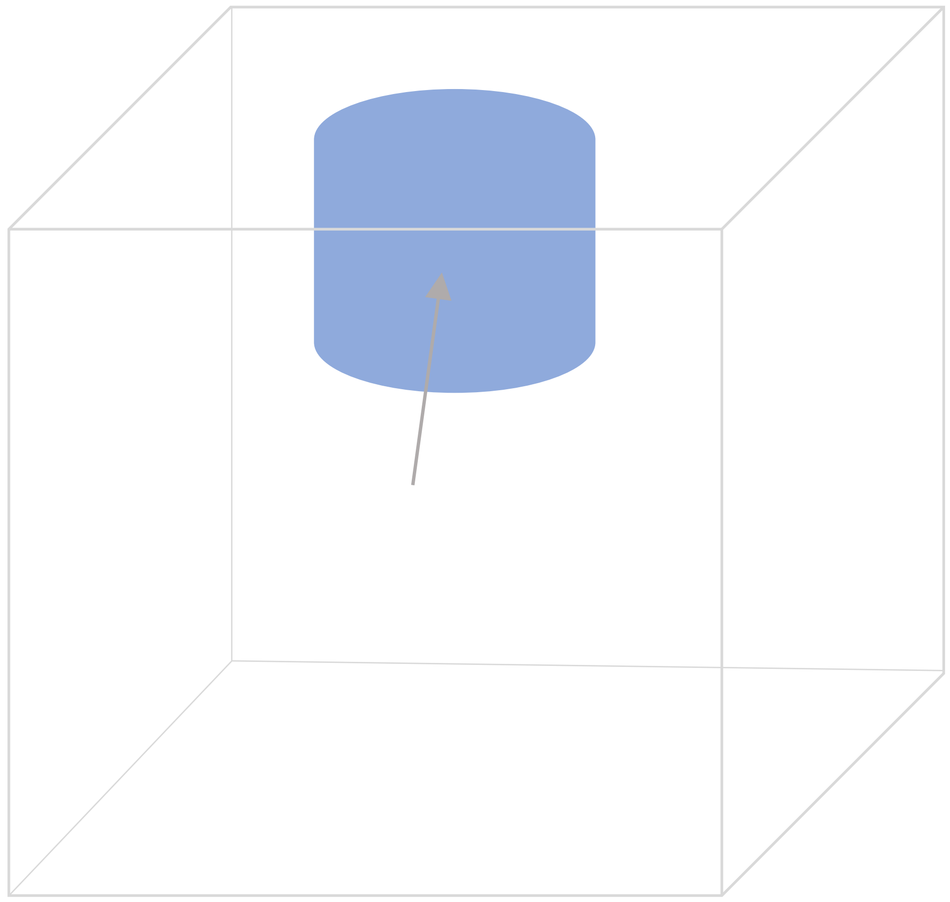
- In Geant4, the geometric model of a virtual experiments consists of one or several **volumes**
- For each volume, Geant4 considered 3 aspects

Aspects of geometry



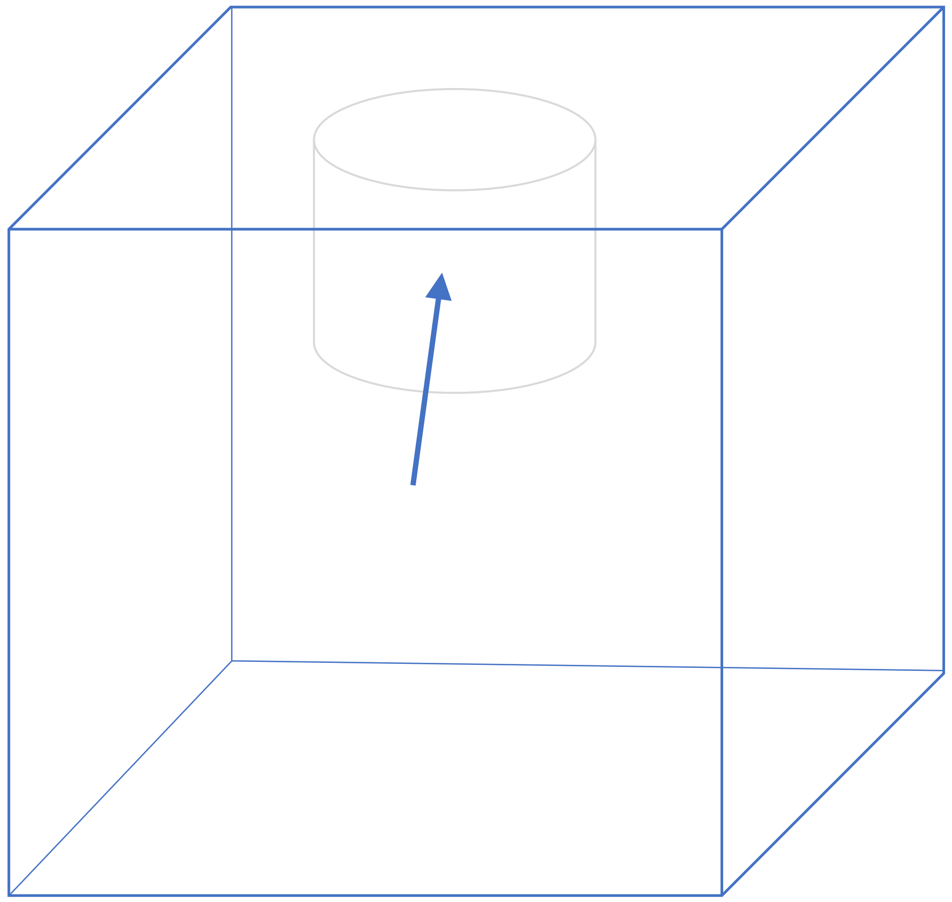
- For each volume, Geant4 considered 3 aspects:
  - The shape and dimensions of the volume is represent by a **solid volume**

Aspects of geometry



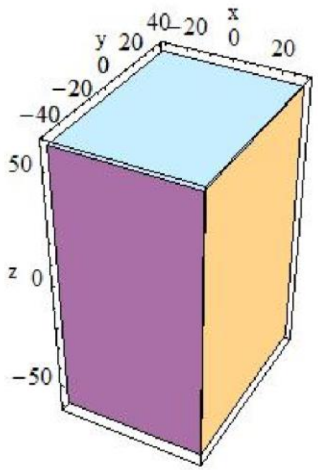
- For each volume, Geant4 considered 3 aspects:
  - The shape and dimensions of the volume is represent by a **solid volume**
  - It is linked to a **material** via the **logical volume**

Aspects of geometry



- For each volume, Geant4 considered 3 aspects:
  - The shape and dimensions of the volume is represent by a **solid volume**
  - It is linked to a **material** via the **logical volume**
  - It is **placed** relative to an **enclosing mother volume** via **physical volume**

Solid Volumes



In the picture:
`pX = 30, pY = 40, pZ = 60`

```
G4Box(const G4String& pName,
             G4double  pX,
             G4double  pY,
             G4double  pZ)
```

- Geant4 provides a set of geometric primitives, the **Constructed Solid Geometry (CSG)** solids, see [BAD, §4.1.2]
- For example, for a cuboid volume use **G4Box**
- It need the **half-length** of the cuboid
- Geant4 understands physical units (e.g. mm, cm, kg, etc.)

Logical Volumes

- Via a `G4LogicalVolume`, a solid volume is linked to a `G4Material`

```
auto* worldVoluem_logic = new G4LogicalVolume(
        worldVolume_solid,//The solid volume belong to the logical volume
        matAir,           //The material associate t
        "world"           //The name of the logical volume;
        );                //for convenient the same as for the solid volume
```

Material Definition

```
//Get a pointer to the manager containing the NIST defined materials
auto* nistMgr = G4NistManager::Instance();
//Get a pointer to the "Air" material; for the names of the materials
//see https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDe
auto* matAir = nistMgr->FindOrBuildMaterial("G4_AIR");
```

- A G4Material can be either manually defined or retrieve from the **`G4NistManager`**
  - Based on data from the *National Institute of Standard And Technology* (NIST) of the US government
  - Available materials are listed in [BAD, §11.6]
    → in this lecture we will use these predefined materials

Physical Placements

```
//3) The "physical volume" rotates and places the logical volume at some
//   point within an enclosing "mother volume"; if no mother volume
//   is given, like here, this volume is defined as the mother volume
//   itself, i.e. the outer most volume
auto* worldVolume_physic = new G4PVPlacement(
        nullptr,          //No rotation
        G4ThreeVector(),  //Placed at (0,0,0)m; the default value of G4ThreeVector
        "world",          //Name of the physical volume
        worldVoluem_logic,//The logical volume that is placed
        nullptr,          //No mother volume because this is the mother volume
        false,            //No Boolean operation
        0,                //Copy number
        checkOverlaps     //Check for overlapping volumes
        );
```

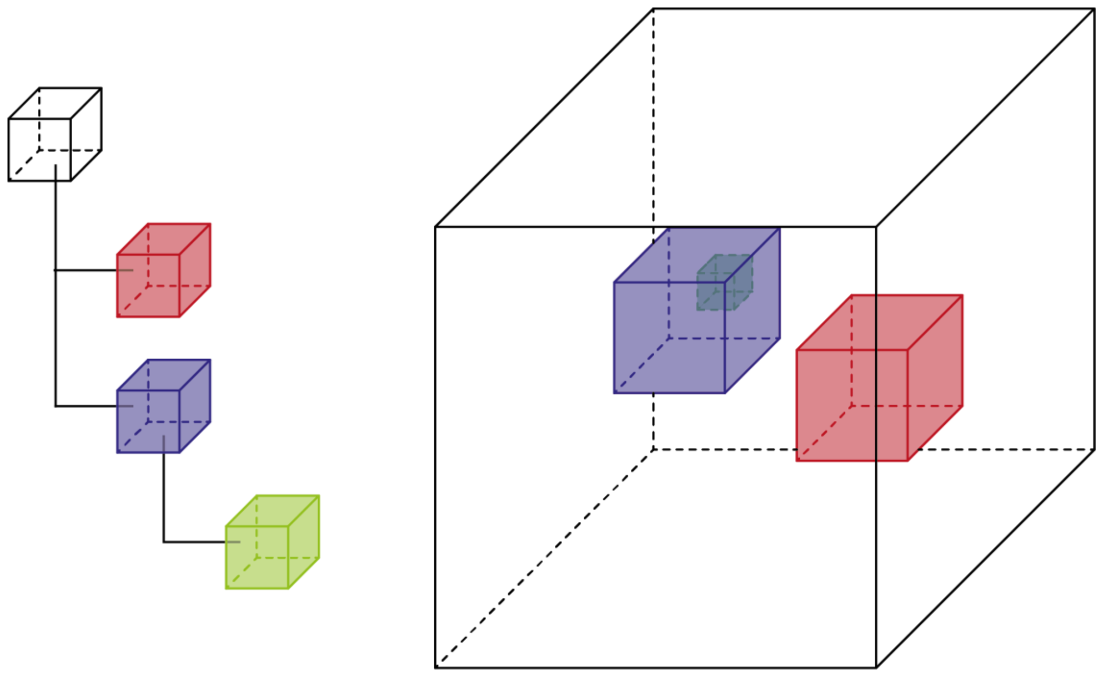
- A `G4VPhysicalVolume` can be created from a logical volume via `G4PVPlacement` constructor
- Geant4 keeps track of volume objects and delete them a the end of a run
  → ***<u>Do not delete them</u>*** in e.g. the destructor
- During development/debugging it is useful to set `checkOverlap=true`
  → checks if volumes which are not mother/daughter occupy the same space

Nested Volumes

```
//3) The "physical volume" rotates and places the logical volume at some
//   point within an enclosing "mother volume"; if no mother volume
//   is given, like here, this volume is defined as the mother volume
//   itself, i.e. the outer most volume
auto* worldVolume_physic = new G4PVPlacement(
        nullptr,          //No rotation
        G4ThreeVector(),  //Placed at (0,0,0)m; the default value of G4ThreeVector
        "world",          //Name of the physical volume
        worldVoluem_logic,//The logical volume that is placed
        nullptr,          //No mother volume because this is the mother volume
        false,            //No Boolean operation
        0,                //Copy number
        checkOverlaps     //Check for overlapping volumes
        );
```

- A volume is placed and rotated **relative** to its **enclosing mother volume**
  → hierarchy of **nested volumes**
- Outermost volume, i.e. those without a mother volume, is the **world volume**
- Construct has to return a pointer to this world volume

Nested Volumes



- A volume is placed and rotated **relative** to its **enclosing mother volume**
  → hierarchy of **nested volumes**
- For example: to model an air-filled iron box, place a smaller, air-filled `G4Box` as daughter volume inside a bigger, iron-filled `G4Box` as mother volume

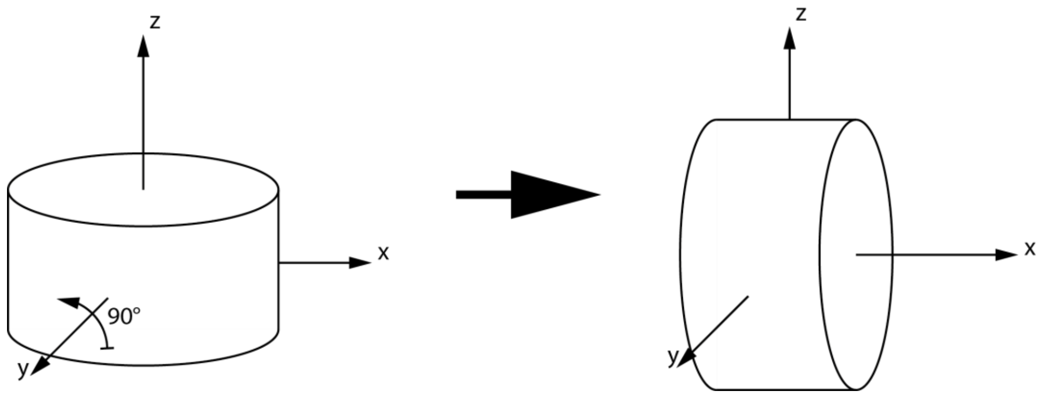
Translation and Rotation

```
G4ThreeVector myTrans =
G4ThreeVector(
 1.*mm,
 -10.3*m,
 3.33*cm
);


G4RotationMatrix *myRot = new
G4RotationMatrix();


new G4PVPlacement(myRot,
myTrans, “myName”, …);
```

- The **translation** of a daughter volume relative to its mother volume is specified via a **G4ThreeVector** object
  - Default value is (0, 0, 0)
- The **rotation** is given via a instance of **G4RotationMatrix**
  - **Do not delete** the matrix after you pass it to the `G4PVPlacement`
  - Delete it in the destructor of `DetectorConstruction`

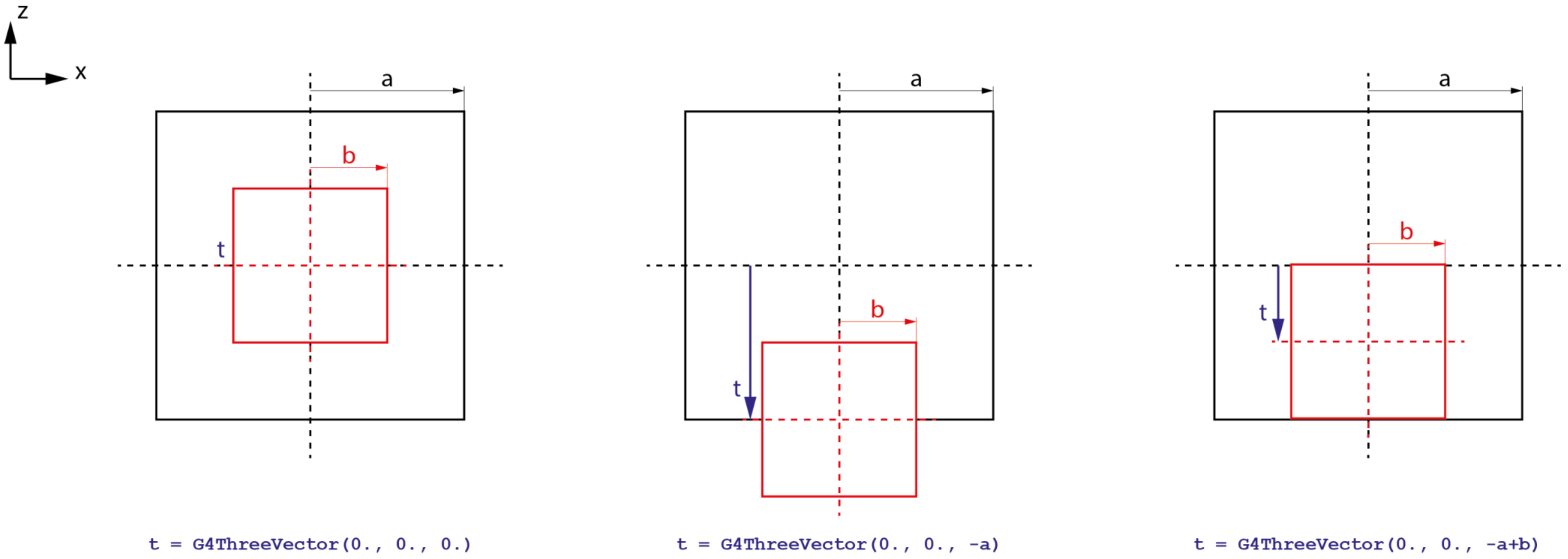
Rotation

```
G4RotationMatrix *myRot = new
G4RotationMatrix();
myRot->rotateY(90.*degree)


new G4PVPlacement(myRot,
myTrans, "myName", …);
```



- G4RotationMatrix is a typedef to CLHEP::HepRotation
- User can define a rotation in various ways, see the [API documentation]
- For example: by default the height of a G4Tubs is aligned to the z-axis, to place it „on the side“ parallel to the x-axis, one can use `rotateY(90.*degree)`

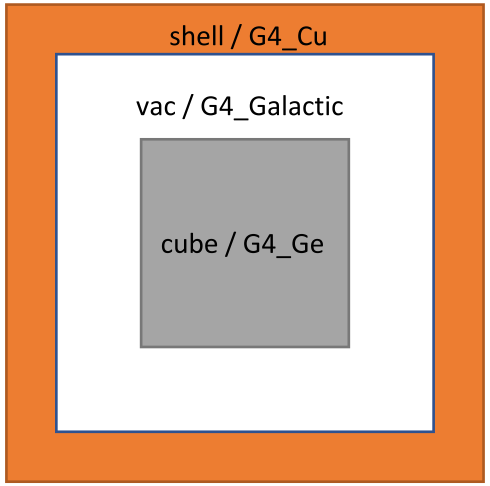
Translation



- Translation **t** is given relative to the centres of **mother** and **daughter** volumes
- By default, the daughter volume is centred with respect to the mother volume
  `t = G4ThreeVector(0., 0., 0.)`

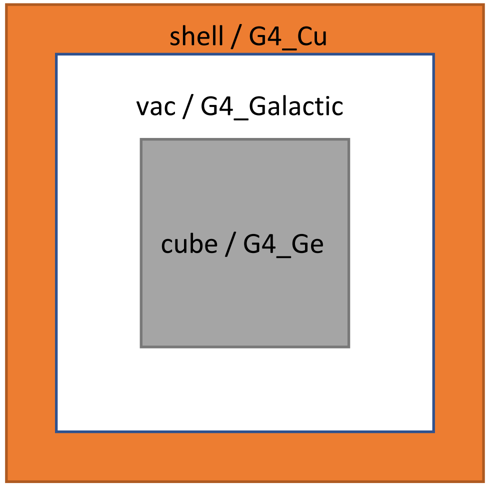
Hands-on

- Open ./src/detectorConstruction.cc in VSC and
  - Change the „PMMA cube“ (lines 78-104) to a cube of
    - 10 cm edge length (caution: G4Box takes *half* edge length as argument)
    - Made of „G4_Ge“ from the NIST material manager
  - Nest „cube“ as daughter volume within a new G4Box with
    - 20 cm edge length
    - Made of „G4_Galactic“ from the NIST material manager
    - Named „vac“
    - „cube“ is placed at the centre (0,0,0) of „vac“
  - Nest „vac“ as daughter volume within a new G4Box with
    - 22 cm edge length
    - Made of „G4_Cu“ from the NIST material manager
    - Named „shell“
    - „vac“ is placed at the centre of „shell“
    - „shell“ is placed at the centre of „world“ (which already exist)
  - Use the modified ./mac/vis.mac from previous hands-on to visualise the setup with JAS3
  - Check that the visualized geometry is correct



Hands-on

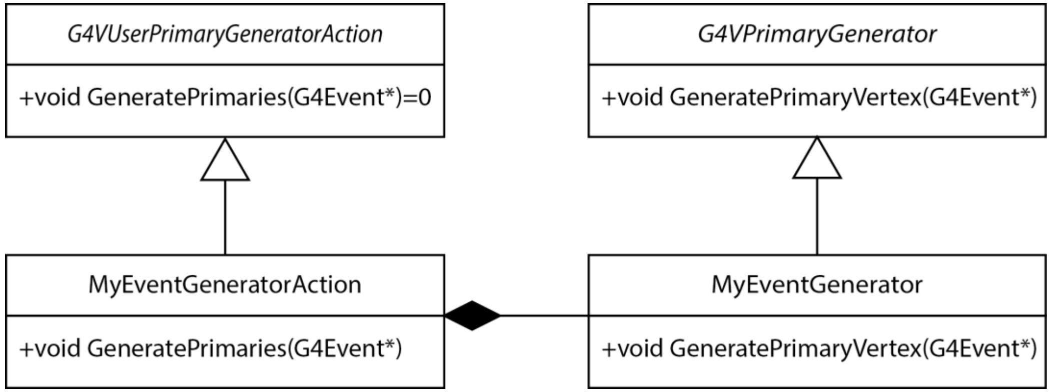
```
    //Place a 22cm³ cube of Cu within the word volume
    auto* matCu = nistMgr->FindOrBuildMaterial("G4_Cu");
    G4double shellHalfLength = 22*cm/2.;
    auto* shell_solid = new G4Box("shell", shellHalfLength, shellHalfLength, shellHalfLength);
    auto* shell_logic = new G4LogicalVolume(shell_solid, matCu, "shell");
    new G4PVPlacement(nullptr, G4ThreeVector(), shell_logic, "shell", worldVoluem_logic, false, 0, checkOverlaps);

    //Place a 20cm³ cube of vacuum within the Cu volume
    auto* matVac = nistMgr->FindOrBuildMaterial("G4_Galactic");
    G4double vacHalfLength = 20*cm/2.;
    auto* vac_solid = new G4Box("vac", vacHalfLength, vacHalfLength, vacHalfLength);
    auto* vac_logic = new G4LogicalVolume(vac_solid, matVac, "vac");
    new G4PVPlacement(nullptr, G4ThreeVector(), vac_logic, "vac", shell_logic, false, 0, checkOverlaps);

    //Place a 10cm³ cube of Ge within the vacuum volume
    auto* matGe = nistMgr->FindOrBuildMaterial("G4_Ge");
    G4double cubeHalfLength = 10*cm/2.;
    auto* cube_solid = new G4Box("cube", cubeHalfLength, cubeHalfLength, cubeHalfLength);
    auto* cube_logic = new G4LogicalVolume(cube_solid, matGe, "cube");
    new G4PVPlacement(nullptr, G4ThreeVector(), cube_logic, "cube", vac_logic, false, 0, checkOverlaps);

    return worldVolume_physic;
}
```



Primary Particle Generation

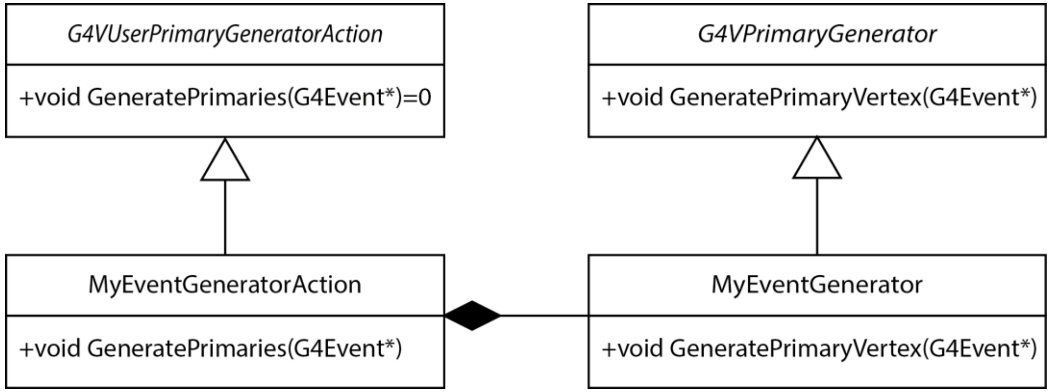
G4VUserPrimaryGeneratorAction; G4VPrimaryGenerator; G4GeneralParticleSource

Class Diagram



- The Geant4 class that implement the generation of a primary particle is the **primary particle generator**
- It is derived from the abstract base class `G4VPrimaryGenerator`
- It has to implement the method `void GeneratePrimaryVertex(G4Event* anEvent)`

Class Diagram

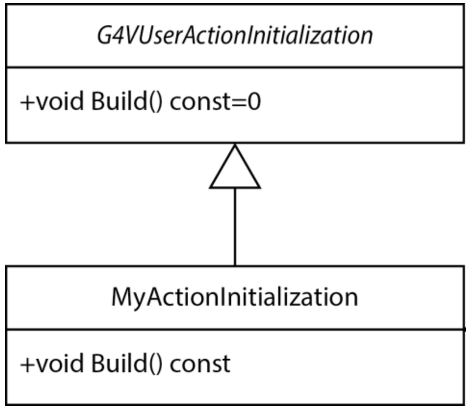


- The generator is instantiate by the **primary generator action**
- It is derived from the abstract base class `G4VUserPrimaryGeneratorAction`
- It has to implement the method `void GeneratePrimaries(G4Event* anEvent)`

G4VUserPrimaryGeneratorAction

src > actionInitialiser.cc > ...

```cpp

void G4minWE::ActionInitialiser::Build() const {
    //Set primary particle generator
    SetUserAction(new G4minWE::PrimaryParticleAction);
    //Set run action
    SetUserAction(new G4minWE::RunAction);
    //Set event action
    SetUserAction(new G4minWE::EventAction);
}
```



- The primary generator action can be instantiate via a dedicated **`G4UserAction Initialization`** class which will handle the registering with `G4RunManager`

G4VUserPrimaryGeneratorAction

```cpp
#ifndef INCLUDE_PRIMARYPARTICLEACTION_HH_
#define INCLUDE_PRIMARYPARTICLEACTION_HH_

#include "G4VUserPrimaryGeneratorAction.hh"
class G4Event;
class G4GeneralParticleSource;

namespace G4minWE{

class PrimaryParticleAction : public G4VUserPrimaryGeneratorAction{

public:
    PrimaryParticleAction();
    ~PrimaryParticleAction() override;

    void GeneratePrimaries(G4Event*) override;

private:
    G4GeneralParticleSource* gps {nullptr};
};
}

#endif /* INCLUDE_PRIMARYPARTICLEACTION_HH_ */
```

- The class itself can be very simple: it just has to instantiate the primary particle generator
- Geant4 provides some predefined primary particle generators:
  - `G4ParticleGun` – to model a vertex with fixed properties
    → Example in G4minWE
  - `G4GeneralParticleSource` (GPS) – can also model more complex scenarios (primary particle homogeneously distributed in a given volume)
    → We'll use it in the hands-on

G4GeneralParticleSource

```cpp
#include "primaryParticleAction.hh"
#include "G4GeneralParticleSource.hh"
#include "G4ParticleTable.hh"
#include "G4SystemOfUnits.hh"
#include "G4Event.hh"

G4minWE::PrimaryParticleAction::PrimaryParticleAction() {
    //Create a "particle gun" that shoot one particle during each event
    gps = new G4GeneralParticleSource();
}

G4minWE::PrimaryParticleAction::~PrimaryParticleAction() {
    delete gps;
}

void G4minWE::PrimaryParticleAction::GeneratePrimaries(G4Event* evt) {
    //This method is called by Geant4 at the beginning of each
    //event: it will create the vertex of the primary particle
    gps->GeneratePrimaryVertex(evt);
}
```

- For the `G4GeneralParticleSource`, the user has to provide very little code, but …
- It is very flexible
- It is controllable via macro commands

G4GeneralParticleSource

```
/run/initialize
```

- First, need to initialize Geant4

G4GeneralParticleSource

```
/run/initialize

/gps/particle e-
```

- First, need to initialize Geant4
- Select the type of particle to be generated
  - Either elementary particle

G4GeneralParticleSource

```
/run/initialize


/gps/particle ion
/gps/ion 1 3
```

- First, need to initialize Geant4
- Select the type of particle to be generated
  - Either elementary particle
  - Or ion ${}^{A}_{Z}X$, e.g. ${}^{3}_{1}H$

G4GeneralParticleSource

```
/run/initialize

/gps/particle ion
/gps/ion 1 3

/gps/energy 1. MeV
```

- First, need to initialize Geant4
- Select the type of particle to be generated
- Kinetic energy at start

G4GeneralParticleSource

```
/run/initialize

/gps/particle ion
/gps/ion 1 3

/gps/energy 1. MeV
/gps/position 0. 0. 0. mm
```

- First, need to initialize Geant4
- Select the type of particle to be generated
- Kinetic energy at start
- Position of source (3D vector *with* units)

G4GeneralParticleSource

```
/run/initialize


/gps/particle ion
/gps/ion 1 3


/gps/energy 1. MeV
/gps/position 0. 0. 0. mm
/gps/direction 1 2 3
```

- First, need to initialize Geant4
- Select the type of particle to be generated
- Kinetic energy at start
- Position at start
- Direction at start (3D vector *without* units, does not need to be a unit vector)

G4GeneralParticleSource

```
/run/initialize

/gps/particle ion
/gps/ion 1 3

/gps/energy 1. MeV
/gps/position 0. 0. 0. mm
/gps/direction 1 2 3

/run/beam 2
```

- First, need to initialize Geant4
- Select the type of particle to be generated
- Kinetic energy at start
- Position at start
- Direction at start
- Start simulation with 2 events

G4GeneralParticleSource

```
/run/initialize
/gps/pos/type Volume
/gps/pos/shape Para
/gps/pos/halfx 1. cm
/gps/pos/halfy 1. cm
/gps/pos/halfz 1. cm
/gps/pos/paralp 0
/gps/pos/parthe 0
/gps/pos/parphi 0
/gps/pos/centre 0. 0. 0. mm
/gps/confine cube
/gps/particle ion
/gps/ion 1 3
/gps/ang/type iso
/run/beam 2
```

- Can be more complex, e.g.
  - Define a cube (=parallelepiped with all angles set to 0)
  - With 1 cm edge length
  - At position (0, 0, 0) mm
  - Filled with ${}^{3}_{1}H$ ions
  - That is confined to the volume “cube”
  - And directions that are isotropic distributed
- Full list of GPS commands

Time Normalisation

- As each simulated event is **independent from each other**, the simulation has **no intrinsic time scale**, i.e. does not “know” how much time is passed between the events

➔“ How long” does the virtual experiment run?

Time Normalisation

- $N_0 = 10^6; A = 100\ \mathrm{kBq}$

$$T = \frac{N_0}{A}$$

$$= \frac{10^6}{100\ \mathrm{kBq}}$$

$$= \frac{10^6}{100 \cdot 10^3\ \mathrm{s}^{-1}}$$

$$= 10\ \mathrm{s}$$

- We need to normalize the amount of simulated events to a known rate, e.g.
  - We model the measurement of a ^{60}Co source with a HPGe detector
  - We simulate N_0=1e6 events, each starts with a ^{60}Co decay
  - The source has an activity of A=100 kBq (1 Bq = 1 decay per second)

➔ In reality, our experiment would have run for 10 s

Time Normalisation

- $N = 10^4; T = 10\ s$

$$R = \frac{N}{T}$$
$$= 10^3\ \mathrm{s}^{-1}$$
$$= \frac{N}{N_0} \cdot A$$

- In the simulation, in *N*=1e4 events an energy above the detection threshold was deposited in the HPGe detector
  - Detection efficiency N/N_0=1%
  - What count rate *R* would this correspond to?

➔ In reality, the HPGe would have a count rate of $R=10^3\ \mathrm{s}^{-1}$*

*Proper unit is s^{-1}, ***not*** Hz; albeit the dimensions are the same, Hz is used for *periodic* events

Hands-on

- Open ./src/primaryParticleAction.cc and the corresponding header file in VSC
- Change the primary particle generator from „G4ParticleGun“ to „G4GeneralParticleSource“
- Modify ./mac/vis_run.mac
  - To use JAS3 for visualisation
  - Use GPS to place ^{71}Ge inside the „cube“ volume
  - Simulate 20 events
- Open the scene-0.heprep.zip file in JAS3: what could the green lines be?

Hands-on

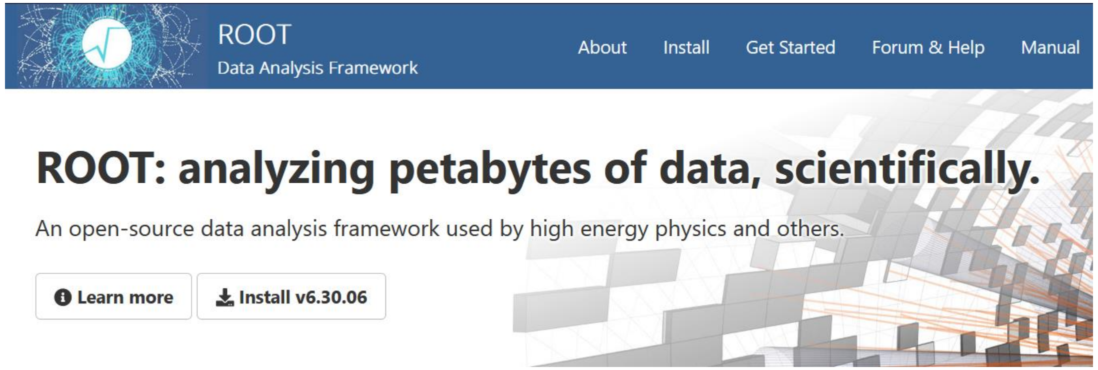
run_vis.mac

```
#Place 71Ge ions
/gps/pos/type Volume
/gps/pos/shape Para
/gps/pos/halfx 5.5 cm
/gps/pos/halfy 5.5 cm
/gps/pos/halfz 5.5 cm
/gps/pos/paralp 0
/gps/pos/parthe 0
/gps/pos/parphi 0
/gps/pos/centre 0. 0. 0. mm
/gps/pos/confine cube
/gps/particle ion
/gps/ion 32 71
/gps/energy 0 MeV
/gps/ang/type iso


#Simulate 10 events, one primary particle per event
/run/beamOn 10
```

Particle Tracking & Data Storage

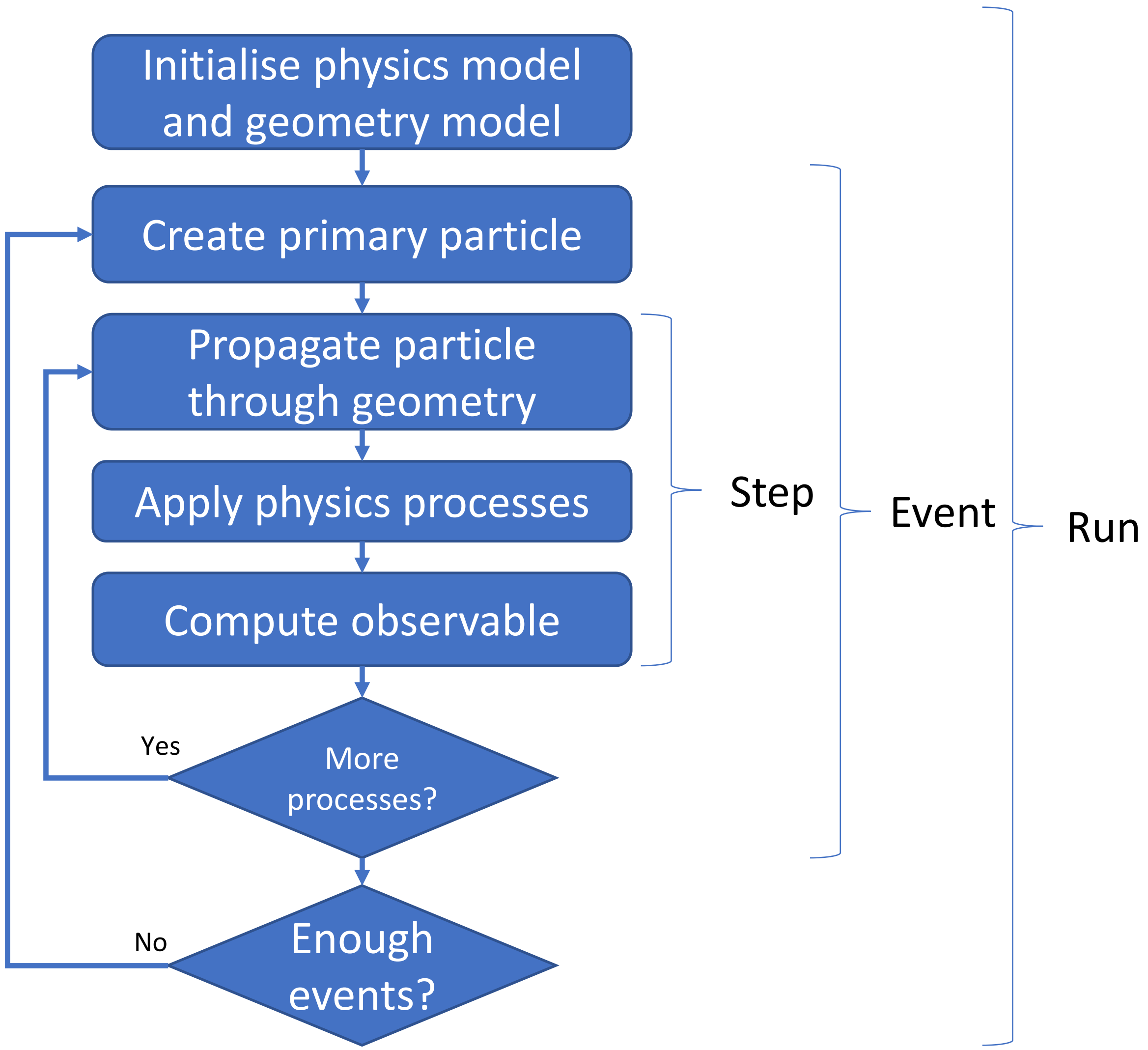
User Action Classes; Run; Event; ROOT

ROOT



- **ROOT** is a data analysis framework developed by CERN and widely used with (high energy) particle physics experiments
  → that's why we will use it
- It's open source: https://root.cern.ch (we will use version 6.22)
- Well documented: https://root.cern/doc/v622
- Generally, data can also be analysed with R, python, Mathlab, etc.

Reminder From Lecture 1



- **Run**: all samples drawn within this particular simulation
- **Event**: one drawn sample
- **Track**: trajectory of one particle (there may be several in one event)
- **Step**: move the particle along the minimal mean free path along its track

User Action Classes

`G4UserRunAction`

`G4UserEventAction`

`G4UserTrackingAction`

`G4UserSteppingAction`

- Geant4 offers 5 optional **User Action classes** [BAD §6.3]
- Deviating these classes, users can
  - Modify the simulation
  - Collect data
- At run/event/track/step level

User Action Classes

G4UserRunAction

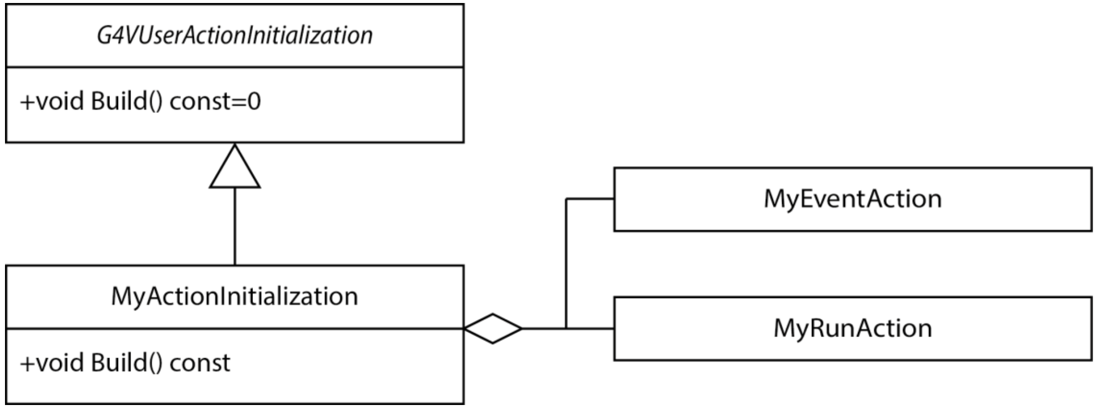
G4UserEventAction

G4UserTrackingAction

G4UserSteppingAction

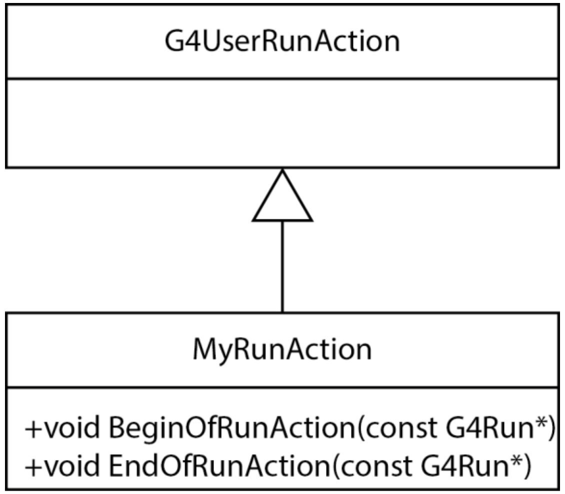
- Geant4 offers 5 optional **User Action classes** [BAD §6.3]
- Deviating these classes, users can
  - Modify the simulation
  - **Collect data**
- At **run/event**/track/step level

Register User Action Classes



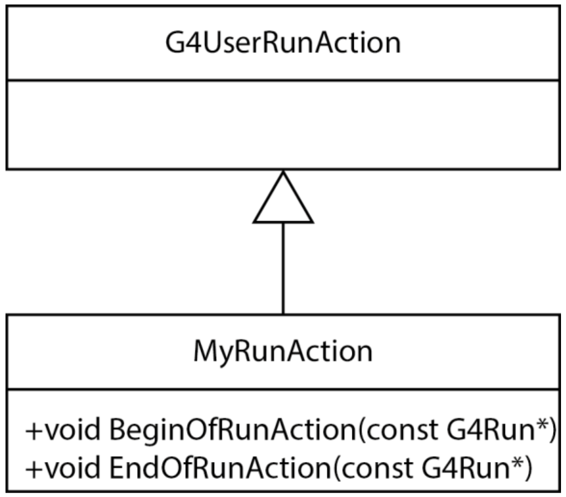
- Like the PrimaryParticleAction, the UserAction are instantiate via a the **`G4UserAction Initialization`** which will handle the registering with `G4RunManager`

G4UserRunAction



- Geant4 provide fully implemented User Run Action class
  → one doesn't *have* to implement it
- But if one provide a deviated subclass, one can **customize many aspects of Geant4's handling of a run**

G4UserRunAction



- For example: by overriding the `BeginOfRunAction` and `EndOfRunAction` methods, one can executed code **before a run starts and after it's finished**
- This way, one could open and close an output files to store the simulated data

G4AnalysisManager

```
#ifndef INCLUDE_RUNACTION_HH_
#define INCLUDE_RUNACTION_HH_

#include "G4UserRunAction.hh"
class G4Run;
class G4RootAnalysisManager;

namespace G4minWE{
class RunAction : public G4UserRunAction{

public:
    RunAction();
    ~RunAction() override = default;

    void BeginOfRunAction(const G4Run*) override;
    void   EndOfRunAction(const G4Run*) override;

private:
    G4RootAnalysisManager* anaMgr{nullptr};

};
}
```

- Geant4 provides predefined manager classes [BAD §9.2] to handle **data storage** as
  - CSV
  - HDF5
  - XML
  - ROOT
- For example, use it to open a **ROOT** output file in the Run Action

G4AnalysisManager

```cpp
G4minWE::RunAction::RunAction() {
    //Get instance of the analysis manager, because we include
    //g4root.hh we will get a G4RootAnalysisManager
    anaMgr = G4AnalysisManager::Instance();
```

```cpp
void G4minWE::RunAction::BeginOfRunAction(const G4Run*) {
    //Open the output file
    G4String fileName = "cube.root";
    anaMgr->OpenFile(fileName);
}

void G4minWE::RunAction::EndOfRunAction(const G4Run*) {
    //Write data to file
    anaMgr->Write();
    //Close file
    anaMgr->CloseFile();
}
```

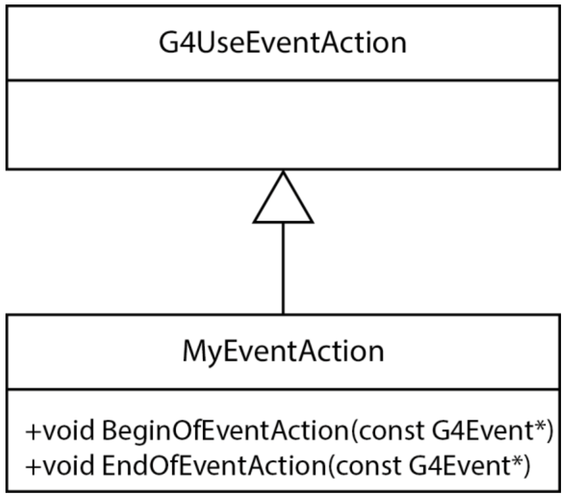
- Geant4 provides predefined manager classes [BAD §9.2] to handle **data storage** as
  - CSV
  - HDF5
  - XML
  - ROOT
- For example, use it to open a **ROOT** output file in the Run Action

ROOT File Structure

```cpp
G4minWE::RunAction::RunAction() {
    //Get instance of the analysis manager, because we include
    //g4root.hh we will get a G4RootAnalysisManager
    anaMgr = G4AnalysisManager::Instance();
    //Create Ntuple
    anaMgr->CreateNtuple(
            "cube",             //Name of the Ntuple
            "Data from cube SD" //Description of the Ntuple
            );
    //Create a column of doubles
    anaMgr->CreateNtupleDColumn("Edep");
    anaMgr->CreateNtupleDColumn("PosX");
    anaMgr->CreateNtupleDColumn("PosY");
    anaMgr->CreateNtupleDColumn("PosZ");
    //Finalize the Ntuple
    anaMgr->FinishNtuple();
    //Create 1D histogram
    anaMgr->CreateH1(
            "cube_Edep",                     //Name of the histogram
            "Energy deposition in cube CS",  //Title of the histogram
            1000,                            //1000 bins ...
            0.,                              //between 0 ...
            100.*keV                         //and 100 keV
            );
}
```

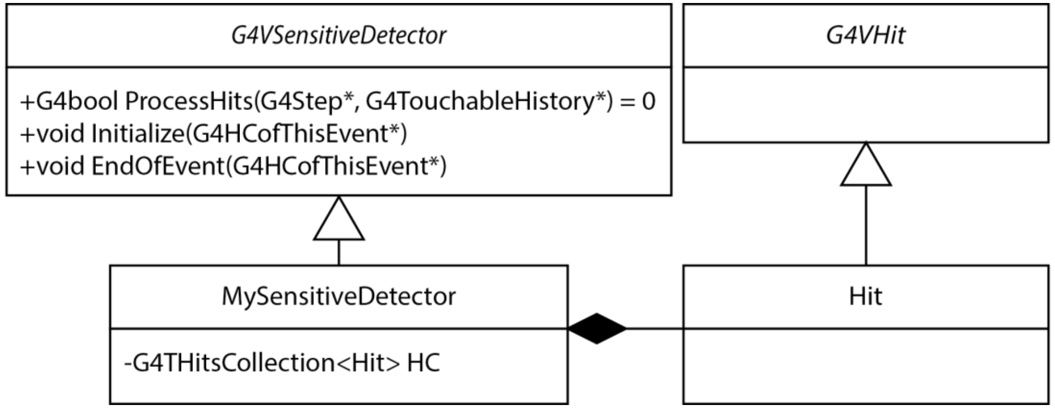
- In the simplest case, a ROOT file structured data sets using a “Table” metaphor:
  - Table ~ **N-tuple** ~ Tree
  - **columns** (of a given data type like int, double, …)
  - Entries ~ **rows**
- In addition, a ROOT file can also contain **histograms** of various dimensions and precisions, e.g. `TH1D` type → 1 dimensional with double precision

G4UserEventAction



- Similarly, deviating a subclass from `G4UserEventAction` allows the **customisation of how Geant4 handles events**
- For example, by overriding the `BeginOfEventAction` and `EndOfEventAction` methods, one **can executed code before an event starts and after it's finished**
- This way, one can perform simple analysis tasks, e.g. extract data from a **sensitive detector**

Sensitive Detector



- To simulate a **particle detector** and the **hits** detected by it, Geant4 provides the abstract base classes `G4VSensitiveDetector` and `G4VHit`, respectively [BAD §4.4]
- The user can deviate a concrete **Sensitive Detector (SD)** and hit classes from it

Sensitive Detector

src > detectorConstruction.cc > ConstructSDandField()

```
void G4minWE::DetectorConstruction::ConstructSDandField() {
    //Define a "sensitive detector" (SD) that can register in
    //principle several quantities
    auto* detector = new G4minWE::SensitiveDetector(
            "cube",     //Name of SD
            "cubeHC"    //Name of hit collection
            );

    //Assign the SD to the logical volume named "cube"
    SetSensitiveDetector(
            "cube",             //Name of logical volume
            detector            //Pointer to SD
            );
    //Add the SD to the SD manager
    G4SDManager::GetSDMpointer()->AddNewDetector(detector);
}
```

- One can **attach** a SD object to a logical volume in `G4VUserDetectorConstruction::ConstructSDandField()`
- Besides user defined SDs, Geant4 provides also general purpose SDs: `G4MultiFunctionalDetector` and `G4VPrimitiveScorer` [BAD § 4.4.4]

Sensitive Detector

```
void G4minWE::SensitiveDetector::Initialize(G4HCofThisEvent *hitCollection) {
    HCollection = new HitsCollection(SensitiveDetectorName, collectionName[0]);

    //Add this collection to hce
    G4int hcID = G4SDManager::GetSDMpointer()->GetCollectionID(
            collectionName[0]);
    hitCollection->AddHitsCollection(hcID, HCollection);

}
```

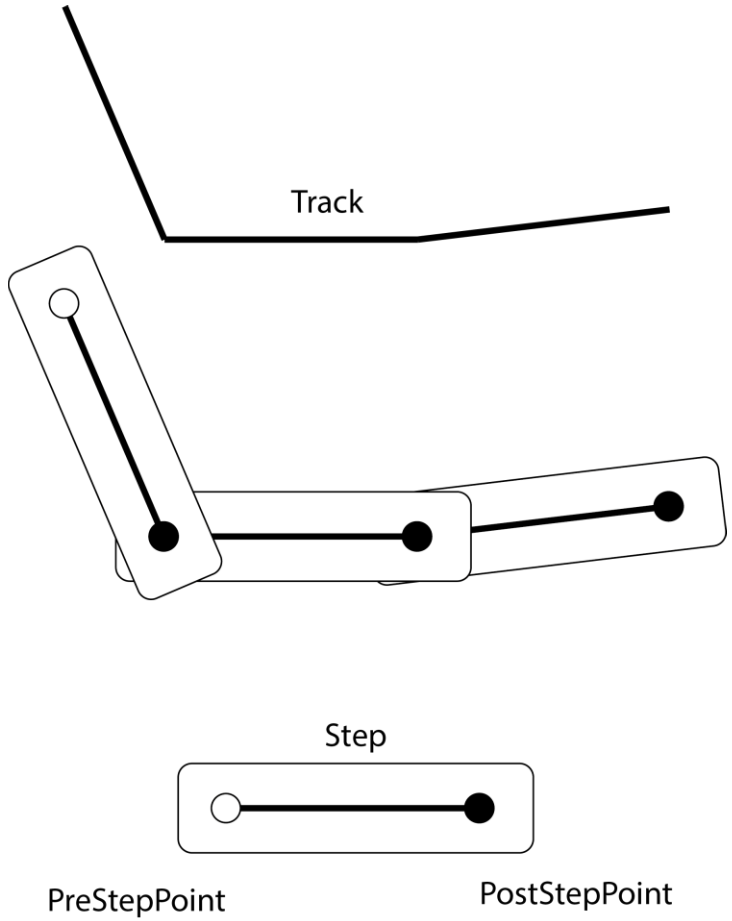
- At the start of each event, the SDs are **initialized**:
  - Each SD initializes a collection to collect future hits = HitCollection (HC)
  - Identified by the SD name and a collection name

Sensitive Detector

```cpp
G4bool G4minWE::SensitiveDetector::ProcessHits(G4Step *step, G4TouchableHistory*) {
    //Get energy deposit by current hit
    G4double edep = step->GetTotalEnergyDeposit();
    //If no energy is deposit, nothing to do
    if (edep == 0.) {
        return false;
    }
    //Otherwise create a new hit
    auto *newHit = new G4minWE::Hit();
    //And set the data
    newHit->SetEnergDeposit(edep);
    newHit->SetPosition(step->GetPostStepPoint()->GetPosition());

    HCollection->insert(newHit);

    return true;
}
```

- Each time a particle track pass through the associated volume, the `ProcessHits()` method of the SD is called
- Data can be accessed through the provided **`G4Step`** object

G4Step



- A `G4Step` is defined as the movement of a `G4Track` between a PreStepPoint and a PostStepPoint and **allows to access the quantities changed** during this move
- Especially, it allows to access
  - The PreStepPoint
  - The PostStepPoint
  - The Track
- Allows access of deposited energy, position and time of hit, see documentation for all available data
- **Default units** are MeV, mm, ns [BAD §3.3]

Sensitive Detector

```
G4bool G4minWE::SensitiveDetector::ProcessHits(G4Step *step, G4TouchableHistory*) {
    //Get energy deposit by current hit
    G4double edep = step->GetTotalEnergyDeposit();
    //If no energy is deposit, nothing to do
    if (edep == 0.) {
        return false;
    }
    //Otherwise create a new hit
    auto *newHit = new G4minWE::Hit();
    //And set the data
    newHit->SetEnergDeposit(edep);
    newHit->SetPosition(step->GetPostStepPoint()->GetPosition());

    HCollection->insert(newHit);

    return true;
}
```

- Each time a particle track pass through the associated volume, the `ProcessHits()` method of the SD is called
- Data can be accessed through the provided **`G4Step`** object
- And filled in **`Hit`** object
- Insert it in the HC
- How to access it, see slide 60

Hits

```cpp
namespace G4minWE{
class Hit: public G4VHit {

public:
    Hit() = default;
    Hit(const Hit&) = default;
    ~Hit() override = default;

    Hit& operator=(const Hit&) = default;
    G4bool operator==(const Hit& right) const;

    inline void* operator new(size_t);
    inline void operator delete(void*);

    void Draw() override;
    void Print() override;

    void SetEnergDeposit(G4double edep);
    void SetPosition(const G4ThreeVector &pos);

    G4double GetEnergyDeposit() const;
    G4ThreeVector GetPosition() const;

private:
    G4double EnergyDeposit { 0. };
    G4ThreeVector Position;
};
```

- The Hit class is mostly a **data container**
- One can fill the hit object with the data accessible from the `G4Step` **object**

Hits

include > hit.hh > {} G4minWE > HitsCollection

```cpp
namespace G4minWE{
using HitsCollection = G4THitsCollection<Hit>;

extern G4ThreadLocal G4Allocator<Hit> *HitAllocator;

inline void* Hit::operator new(size_t) {
    if (!HitAllocator) {
        HitAllocator = new G4Allocator<Hit>;
    }
    return (void*) HitAllocator->MallocSingle();
}

inline void Hit::operator delete(void *hit) {
    HitAllocator->FreeSingle((Hit*) hit);
}
}
```

- The Hit class is mostly a **data container**
- One can fill the hit object with the data accessible from the `G4Step` object
- For optimisation issues, Geant4 **prescribe non-standard memory allocation**
  → Just copy'n'paste it from Geant4 examples and adapt names

Store Hit Data In A ROOT File

```cpp
void G4minWE::EventAction::EndOfEventAction(const G4Event* anEvent) {
    //After the current event is finished, process the "hits" recorded
    //by the scorer "edep" of SD "cube" to get the energy deposited
    //inside "cube"

    //1) Get "hit collection" (hc) of this event, i.e. collection
    //   of _all_ hits recorded during the simulation of the current event
    //   by _all_ SDs
    auto* hce = anEvent->GetHCofThisEvent();
    if(!hce){
        //If a nullptr was returned, then there were no hits collected
        //during the current event. Nothing to do here, so end this method.
        return;
    }

    //2)   Select the hit collection "cubeHC" of SD "cube"
    //2.1) Get the ID of the hit collection "cubeHC"
    G4int id = G4SDManager::GetSDMpointer()->GetCollectionID("cubeHC");
    //2.2) With the ID select the HC
    auto* hitCol = hce->GetHC(id);
    //2.3) Get a vector
    auto* hitVec = static_cast<G4minWE::HitsCollection*>(hitCol)->GetVector();
```

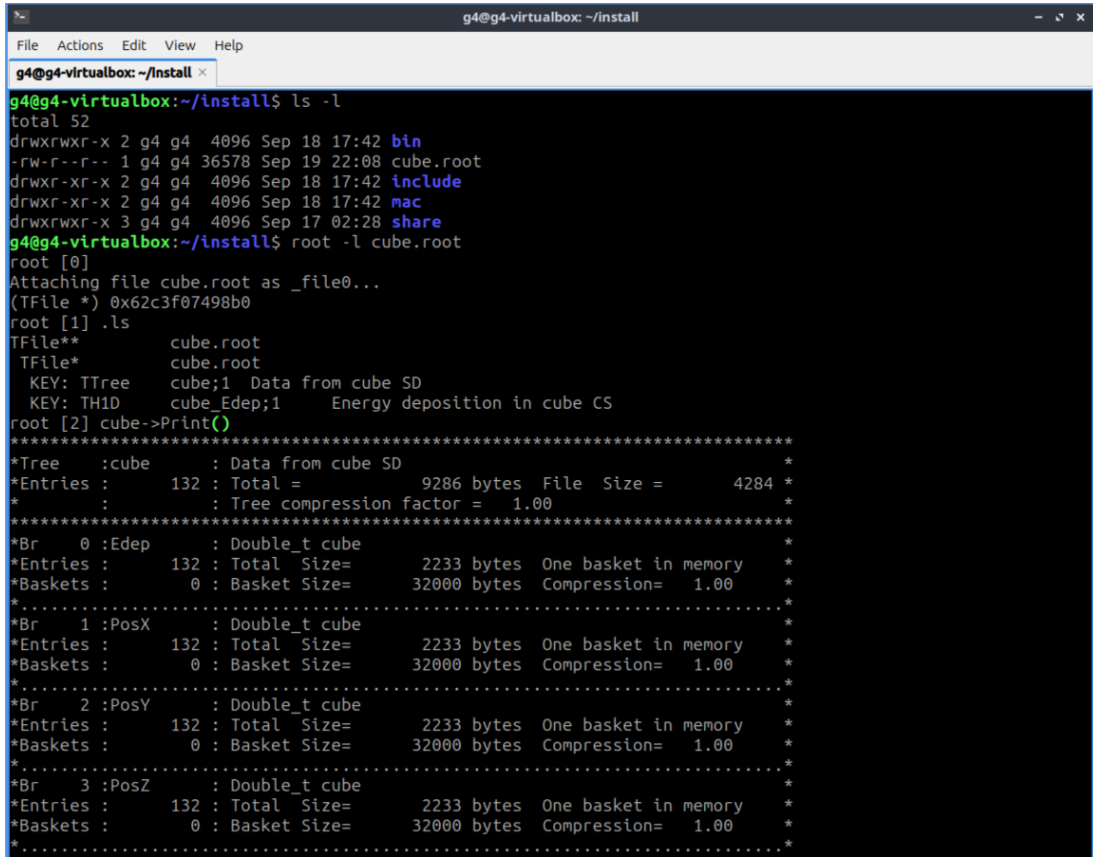
- In `EventAction::EndOfEventAction`
  - **Get the HitsCollection** of the current event
  - Get the hit collection of the SD one is interest in – via look-up the collection name
  - The entries of the collection have the abstract base class `G4VHit` as type
  - For the sake of convenience, cast it to a `std::vector<Hit>` of the actual subclass

Store Hit Data In A ROOT File

```cpp
for (auto* hit : *hitVec){
    //The iterator itr points to an pair, to get the second element, i.e.
    //the value of the pair, do:
    G4double eDep = hit->GetEnergyDeposit();
    const G4ThreeVector& pos = hit->GetPosition();
    //Fill energy into Ntuple and histogram
    //(one has to know that "cube_Edep" histogram was the first
    //created in runAction, i.e. that it has the ID=0; similarly
    //the IDs of posX, posY, posZ are 1, 2, 3, respectively)
    anaMgr->FillH1(
            0,      //ID of the histogram to fill
            eDep    //Value to fill in the histogram
            );
    anaMgr->FillNtupleDColumn(
            0,      //ID of the column to fill
            eDep    //Value to fill in the column
            );
    anaMgr->FillNtupleDColumn(
            1,      //ID of the column to fill
            pos.x()//Value to fill in the column
            );
    anaMgr->FillNtupleDColumn(
            2,      //ID of the column to fill
            pos.y()//Value to fill in the column
            );
    anaMgr->FillNtupleDColumn(
            3,      //ID of the column to fill
            pos.z()//Value to fill in the column
            );
    anaMgr->AddNtupleRow();

}
```

- In `EventAction::EndOfEventAction`
  - **Loop over** all entries of the vector to get the individual hits
  - Read-out the relevant data fields from the hit object
  - Fill the data in the relevant N-tuple / histograms of the ROOT file
  - Once all entries of the N-tuple (=columns) are filled, finalise the N-tuple (=row) by calling `AddNtupleRow()`

ROOT Prompt

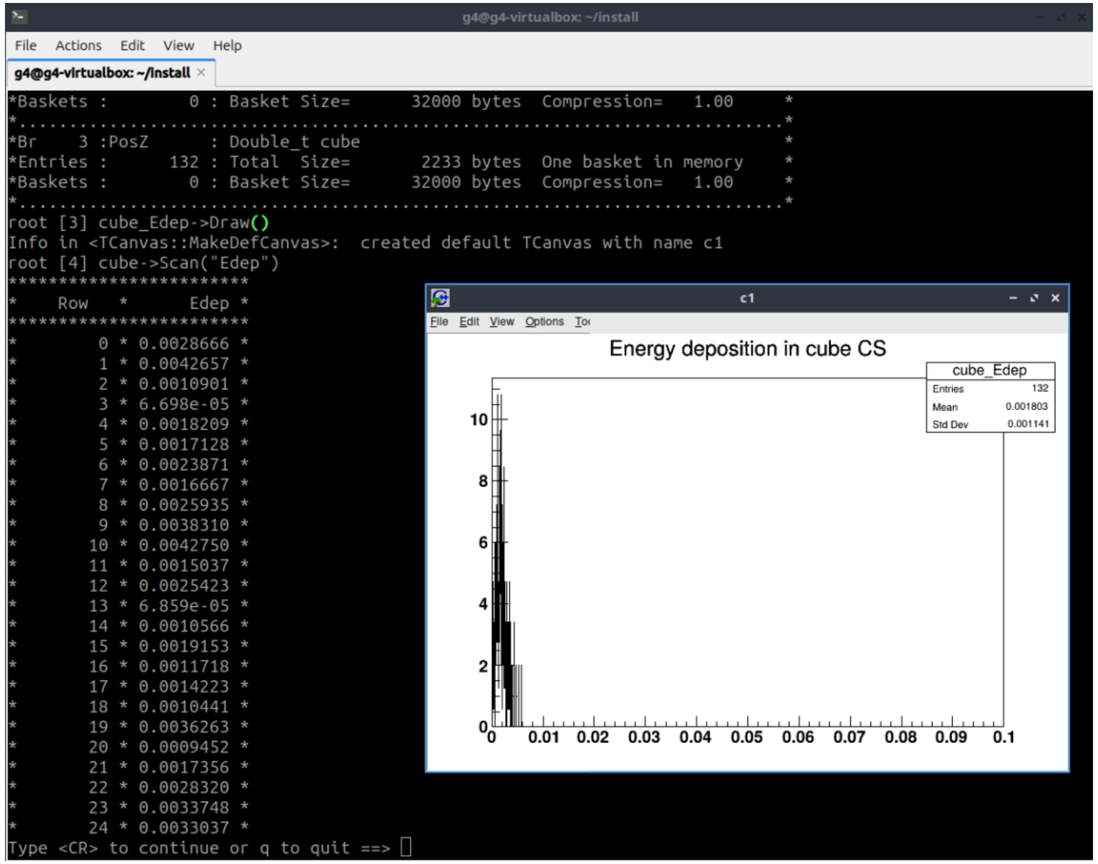


```
g4@g4-virtualbox: ~/install
File  Actions  Edit  View  Help
g4@g4-virtualbox: ~/install
g4@g4-virtualbox:~/install$ ls -l
total 52
drwxrwxr-x 2 g4 g4  4096 Sep 18 17:42 bin
-rw-r--r-- 1 g4 g4 36578 Sep 19 22:08 cube.root
drwxr-xr-x 2 g4 g4  4096 Sep 18 17:42 include
drwxr-xr-x 2 g4 g4  4096 Sep 18 17:42 mac
drwxrwxr-x 3 g4 g4  4096 Sep 17 02:28 share
g4@g4-virtualbox:~/install$ root -l cube.root
root [0]
Attaching file cube.root as _file0...
(TFile *) 0x62c3f07498b0
root [1] .ls
TFile**         cube.root
 TFile*         cube.root
  KEY: TTree    cube;1  Data from cube SD
  KEY: TH1D     cube_Edep;1     Energy deposition in cube CS
root [2] cube->Print()
******************************************************************************
*Tree    :cube      : Data from cube SD                                      *
*Entries :      132 : Total =            9286 bytes  File  Size =       4284 *
*        :          : Tree compression factor =   1.00                       *
******************************************************************************
*Br    0 :Edep      : Double_t cube                                          *
*Entries :      132 : Total  Size=       2233 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    1 :PosX      : Double_t cube                                          *
*Entries :      132 : Total  Size=       2233 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    2 :PosY      : Double_t cube                                          *
*Entries :      132 : Total  Size=       2233 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    3 :PosZ      : Double_t cube                                          *
*Entries :      132 : Total  Size=       2233 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
```

- The **ROOT prompt*** *interprets* C++ commands
- Open a ROOT file via
  `root -l <name of file>`
  (`-l` suppress the splash screen)
- List content of file via
  `.ls`
- List the structure of a TTree via
  `TTree::Print()`
- Close the ROOT prompt via
  `.q`
- For details see Manual

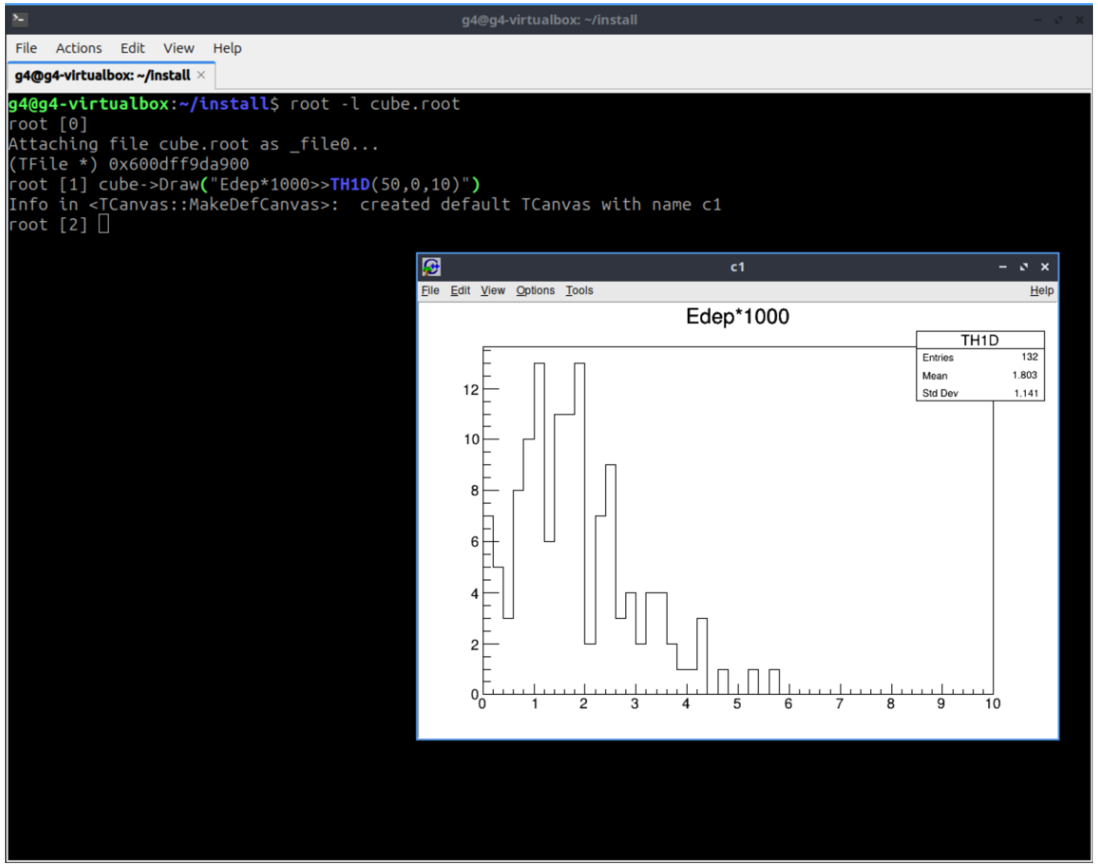
*Alternative ways to interact with ROOT are PyROOT or jupyter notebooks.

Read a ROOT File



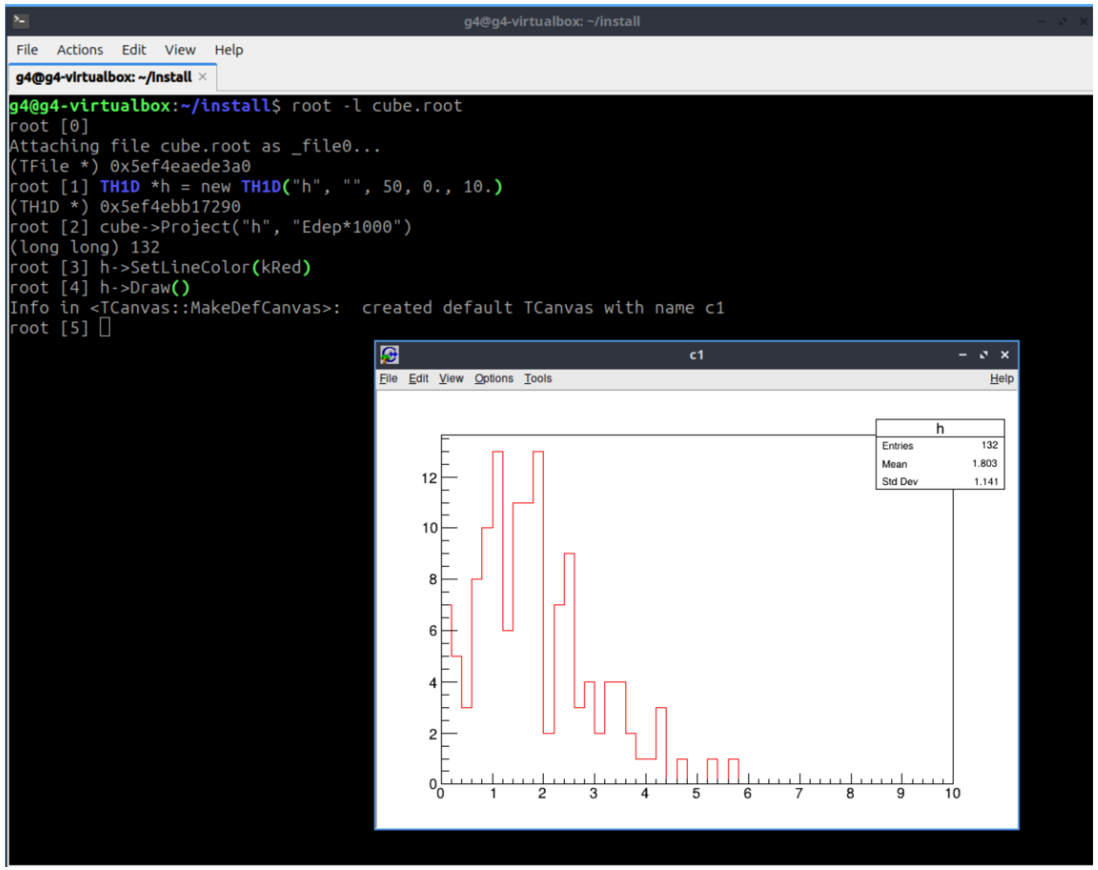
- Open a ROOT file via
  `root -l <name of file>`
  (`-l` suppress the splash screen)
- List content of file via
  `.ls`
- List entries of column Y of tree X
  `X->Scan("Y")`
- Draw histogram Z
  `Z->Draw()`

1-dim Histogram – Draw



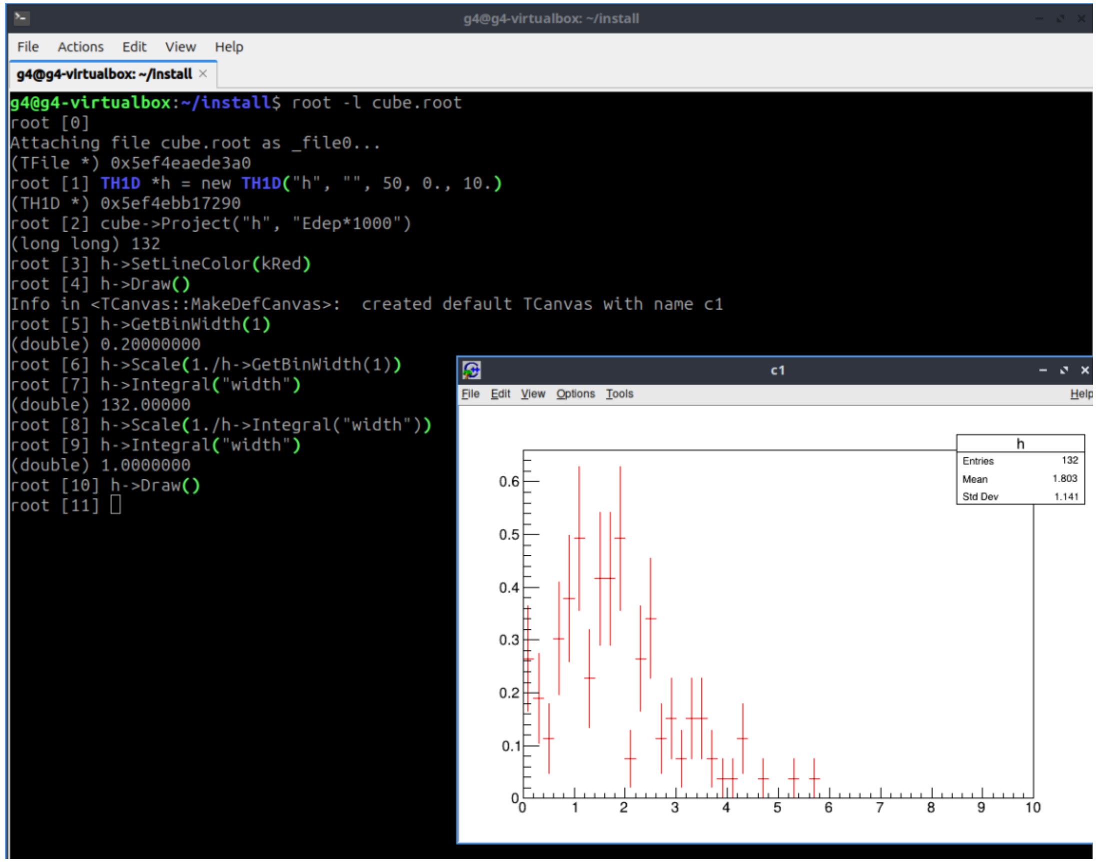
- A histogram can be created in several ways
  - `TTree::Draw(<Expression>, <Cut>, <Options>)`
  - Where `<Expression>`
    - Specifies the column that contains the data to be drawn, here: Edep
    - May contain mathematical operations on this data, here: multiply Edep with 1000 to go from MeV to keV
    - Can specify the type of histogram, here: TH1D with 50 bins between 0 and 10

1-dim Histogram – Project



- A histogram can be created in several ways
  - `TTree::Draw(<Expression>, <Cut>, <Options>)`
  - `TTree::`**`Project`**`(<Histogram name>, <Expression>, <Cut>, <Options>)`
    - Similar, but store the histogram data in an already existing histogram object of name `<Histogram name>` → easier to process it, e.g. modifying line color
    - Create histogram object via, e.g. `TH1D(<Name>, <Title>, <NbBins>, <Min>, <Max>)`

1-dim Histogram - Normalisation



- Normalize it to **bin width**
  - Get bin width of bin 1
    `TH1D::GetBinWidth(1)`
  - Multiply all bins with a given factor a
    `TH1D::Scale(a)`
  - → `TH1D::Scale(1./ TH1D::GetBinWidht(1))`
- To get an empirical **Probability Density Function** (PDF), normalize the histogram to unity
  - **Integral** over the histogram
    `TH1D::Integral("width")`
    To get Integral=$\sum_i y_i \cdot \Delta x_i$
  - →`TH1D::Scale(1./ TH1D::Integral("width"))`

Hands-on

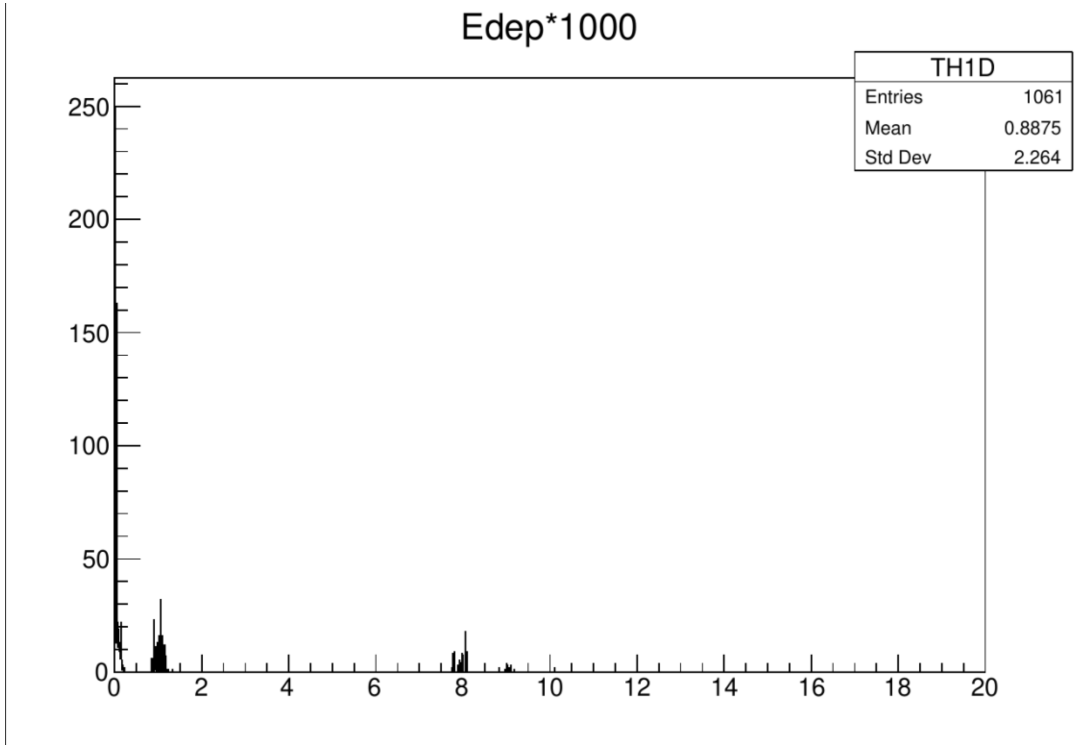
- Adapt ./mac/run.mac to the same GPS settings we used in the previous hands-on and run it for 100 events
- Rename the produced cube.root file to cube1.root, open it in ROOT, use the Draw command to plot “Edep*1000” in the range [0,20], store the plot via “File > Save as”
- Open ./src/RunAction.cc in VSC and delete the DColumns “PosX”, “PosY”, and “PosZ”
- Open ./src/EventAction.cc in VSC,
    - Delete the commands that previously filled the “PosX”, “PosY” and “PosZ” columns
    - Move the commands the filled the “Edep” column and histogram from inside the loop (lines 53-90) to after the loop
    - Add a double variable „sum“ that is Initialized to 0 before the loop
    - Inside the loop, add „eDep“ to sum for each loop iteration
    - After the loop, fill the value of „sum“ to the column „Edep“ and the histogram
- Compile and install the program, run the same macro file as before
- Open the produced root file in ROOT and make the same analysis as before
- What differences do you observe?

Hands-on

Sum energy of Auger electrons	Energy of X-ray	Percent of all decays
10.367	0.0	41.4
1.143	9.224	13.7
1.116	9.251	27.4
0.107	10.260	1.7
0.103	10.264	3.5
1.299	0.0	10.3
0.160	0.0	2.0

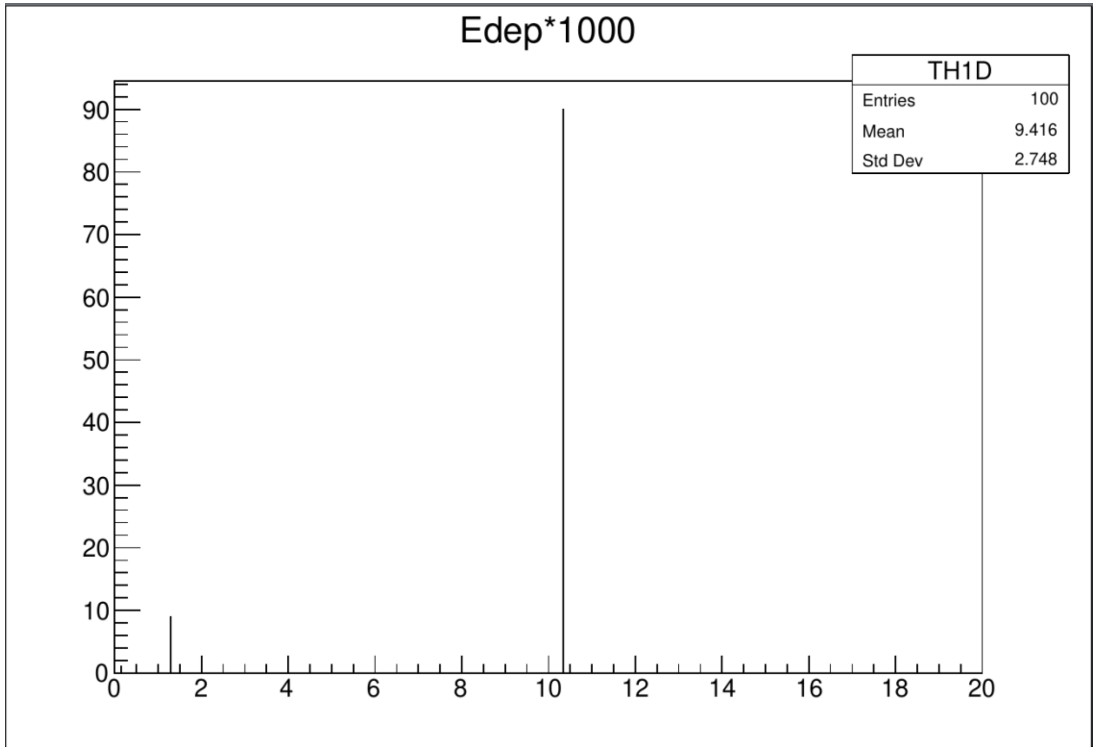
[D.N. Abdurashitov et al., NIM B 373 (2016) 5-9]

Before modification



Energy deposition per „hit“
~ by each e-/X-ray interaction

After modification



Energy deposition per „event“
~ by each ^{71}Ge decay

Take Home Messages

- An accurate simulation needs an accurate geometry
  → in principle not difficult but needs time and good spatial sense
- Before developing your own primary particle generator
  → check if the General Particle Source is sufficient
- Storing of simulated data
  - Choose a file format that‘s supported by your analysis tools
  - Make a deliberate decision what to store: per hit, per event, applying some selection criterion or not, …